

AD-A196 736

DTIC FILE COPY (4)

ULTRAWALL ELECTRONICS ARCHITECTURES

Texas Instruments Incorporated
13500 North Central Expressway
P. O. Box 655936, M.S. 105
Dallas, Texas 75265

13 January 1988

Final Technical Report for period 5 August 1985 - 4 August 1987

Approved for public release; distribution unlimited.

Prepared for

Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

DTIC
ELECTRONIC
JUN 06 1988
S H D

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA196736

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 08-88-32		5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Texas Instruments Incorporated		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research Department of the Navy	
6c. ADDRESS (City, State, and Zip Code) 13500 N. Central Expressway Dallas, Texas 75265		7b. ADDRESS (City, State, and Zip Code) 800 N. Quincy Street Arlington, VA 22217-5000		
8a. NAME OF FUNDING/SPONSORING ORG. (If applicable) Office of Naval Research		8b. OFFICE SYMBOL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-C-0760	
8c. ADDRESS (City, State, and Zip Code) 800 North Quincy Street Arlington, VA 22217-5000		10. SOURCE OF FUNDING NUMBERS		
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Ultrasmall Electronics Architectures				
12. PERSONAL AUTHOR(S) G. A. Frazier				
13a. TYPE OF REPORT Final Technical	13b. TIME COVERED 5 August 1985 - 4 August 1987	14. DATE OF REPORT (Year, Month, Day) 15 January 1988		15. PAGE COUNT 402
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	Nanoelectronics, Chip Architectures, Cellular Automata, Fault Tolerant Circuits	
19. ABSTRACT (Continue on reverse side if necessary and identify by block number)				
<p>Ultrasmall electronics research (nanoelectronics) aims at providing quantum semiconductor devices and device-level architectures that challenge fundamental physical limits. This report summarizes our progress in developing an architectural basis for nanoelectronics. We have determined that the cellular automaton (CA) best satisfies the requirements of nanoelectronic architecture. We quantified the dynamic behavior of CAs and determined useful computational properties of nanometer-sized devices.</p>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Clifford G. Lau		22b. TELEPHONE (Include Area Code) (818) 360 2345	22c. OFFICE SYMBOL	

DD FORM 1473 84 MAR

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. ABSTRACT (Continued)

Computer simulations of multidimensional CA were used to understand the dynamics and computational properties of arrays of locally coupled active devices. We examined large arrays for noise sensitivity. Finite automata theory was used to develop a multivalued algebra for modeling quantum coupled devices.

To accelerate the survey, special-purpose, high-speed simulators were constructed, allowing processing throughputs of 1.2 billion cell operations per second. We wrote a flexible software package for interactive development of CA models.

We determined noise effects in stable CA behaviors, "attractors," and measured the Markov properties of many CAs. Nearest-neighbor automata were grouped according to the size and number of state attractors. The long-term dynamics of these systems are, in general, fault-prone. We examined the sensitivity of two-dimensional, totalistic CA to noise and found that the average state could be controlled by the perimeter cells. Our model displayed transistor-like switch action and fault tolerance of both soft and hard errors. We also examined the ability of locally coupled arrays to store random patterns and demonstrated retrieval using simple correlation techniques. A method for enhancing memory recall was developed. Finally, we devised a simple method for managing clock skew in asynchronous cellular automata.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
I. EXECUTIVE SUMMARY.	1
1. Motivations for the Research.	1
2. Objective of Architecture Research.	3
3. Research Approach	3
4. Summary Of Contract Results	5
4.1 Development of Simulation Environment	5
4.2 Characterization of One-Dimensional Automata.	5
4.3 Characterization of Two-Dimensional Automata.	6
4.4 Development of Quantum Algebra and Circuit Concepts . .	7
4.5 Self-Timed Cellular Automata.	7
II. MOTIVATIONS AND OBJECTIVES OF ARCHITECTURE RESEARCH.	9
1. Introduction to Problem	9
2. The Scaling Limits of Conventional Technology	10
2.1 Transistor Scaling Limits	10
2.2 Connectivity Scaling Limits	10
2.3 Yield Scaling Limits.	11
2.4 Addressability Scaling Limits	11
2.5 Superposition Limits.	12
2.6 Limits of Static Design	13
3. Essentials for a Successful Post-VLSI Era	13
3.1 Discussion.	14
III. NANO-ELECTRONICS.	16
1. Nanoelectronics Technology.	16
1.1 Quantum Coupled Devices	16
1.2 Cellular Automata Architectures	19
1.3 Combining Quantum Coupled Devices and Cellular Automata	21
IV. SUMMARY OF SIMULATION TOOLS DEVELOPED.	24
1. Software Simulation Tools Developed	24
2. Hardware Simulation Tools	25
V. CHARACTERIZATION OF ONE-DIMENSIONAL CELLULAR AUTOMATA.	31
1. Theoretical Development	32
1.1 Basic Dynamics of a Cellular Automaton.	32
1.2 State Attractors as Elements of a Markov Chain.	36

	2. Experiments	38
	3. Markov Results.	39
VI.	MULTIVALUED LOGIC IN QUANTUM COUPLED CIRCUITS.	41
	1. Introduction.	41
	1.1 Advantages of Multivalued Logic Approaches.	42
	2. Theory of Multivalued Algebra	42
	2.1 Functional Decomposition in M-Valued Logic.	44
	2.2 Well Known M-Valued Algebra	48
	2.3 Completeness and Practicality in M-Valued Circuits.	51
	2.4 Overspecification in M-Valued Algebra	53
	3. M-Valued Logic in Quantum Coupled Circuits.	54
	3.1 Navigating in a Quantum World	54
	3.2 Multivalued Quantum Logic	56
	3.3 Ternary Quantum Logic Structures.	56
	4. Additional Notes.	60
	5. Summary	63
VII.	SELF-TIMED CELLULAR AUTOMATA	64
	1. Introduction.	64
	2. Model for Asynchronous Cellular Arrays.	65
	2.1 Experiment.	69
	2.2 Discussion.	69
	REFERENCES	73

LIST OF APPENDICES

- A. Reduced Equations for Nearest-Neighbor Cellular Automata
- B. Space-Time Plots of Nearest-Neighbor Cellular Automata
- C. Attractor Spectra for One-Dimensional Cellular Automata
- D. Attractor Basin Volumes for One-Dimensional Cellular Automata
- E. Sensitivity of One-Dimensional Cellular Automata to Initial Cell Patterns
- F. Markov Matrix Elements for One Dimensional Cellular Automata

LIST OF ILLUSTRATIONS

<u>FIGURE</u>		<u>PAGE</u>
1.	Two-dimensional lattice of a typical cellular automaton. . .	2
2.	Nanoelectronics development path	4
3.	GaAs quantum dots embedded in a sea of AlGaAs.	18
4.	Potential energy diagram for conduction bands of quantum dots	18
5.	Elastic (strong) and inelastic (weak) tunneling between quantum wells.	20
6.	Time evolution of cellular automata.	22
7.	Interwell effects can gate charge transport.	23
8.	Flow diagram of pipelined one-dimensional cellular automata simulator.	27
9.	Flow diagram of high-speed, two-dimensional cellular automata simulator.	28
10.	Flow diagram of fully parallel, one-dimensional cellular automata simulator	29
11.	Ring and torus boundary conditions for next-nearest-neighbor cellular automata.	30
12.	Schematic representation of limit cycle behavior for time irreversible CA.	33
13.	Trajectory classes in cellular automata.	35
14.	The effects of noise on the dynamics of nearest-neighbor coupled, one-dimensional CA.	40
15.	General M-valued, N-input logic gate	43
16.	Logic symbol and function table for a popular (LS181) ALU module	46
17.	Six popular [2,2]-space logic functions.	47
18.	Quantum jumps in charge energy due to quantum size effect. .	57
19.	Construction of "%" operator, using quantum well structures.	61
20.	Construction of "%" operator, using quantum well structures.	62
21.	High-level representation of asynchronous automation cell. .	67
22.	Transition graph for asynchronous automata clock phases and cell operations.	68
23.	Time evolution of a completely self-timed cellular automaton	71

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
1.	Relative Performance of Cellular Automata Simulation Architectures	25
2.	Function Space for Various Numbers of Inputs (N) and Output States (M).	45
3.	[2,2]-Space Operator Set for Conventional Boolean Algebra. .	49
4.	[3,2]-Space Operator Set for Conventional "Min" - "Max" Algebra	49
5.	Truth Table and Canonical Form for Randomly Selected [3,2]-Space Function.	51
6.	Combinatorally Complete Set of Ternary Logic Operators . . .	55
7.	Result of Adding Charge Units Q_1 and Q_2 to Four Types of Quantum Wells	59
8.	Internal Cell Clock Phase vs Phase of Local Cell Neighborhood	70

I. EXECUTIVE SUMMARY

Ultrasmall Electronics Research, now popularly called Nanoelectronics, is a research effort aimed at providing semiconductor devices and device-level architectures that are downscalable to fundamental physical limits. This report summarizes our progress in developing an architectural basis for nanoelectronics under ONR Contract N00014-85-C-0760. In this section, we summarize our research motivations, objectives, approaches, and results. Section II provides an overview of the technological limits driving this research. Section III discusses our overall approach to next-generation nanotechnology. Subsequent sections present our experimental results in detail. Appendices A through F contain the large quantity of reduced data collected over the course of this research.

1. Motivations for the Research

The downscaling of the minimum lateral geometries of transistor-based integrated circuits will eventually be brought to an end by a combination of problems related to devices, interconnection noise, and reliability. The performance limits of conventional integrated circuits are expected to be reached within 20 years. The use of geometric scaledown to further increase the density of on-chip functionality is inhibited by both the inability to scale interconnect isolation and degradation of device properties as local interdevice coupling becomes a dominant effect. Avoiding these limits requires revolutionary approaches to both devices and architectures that exploit the unique properties of nanometer-sized electronic structures. The casualties of this revolution will include high connectivity architectures, transistors, and traditional circuit concepts.

The solution to the scaledown barrier requires:

1. Totally new device concepts which circumvent the scaling limits of conventional transistors; and
2. New, fault-tolerant architectures that support computation without requiring random, long-range connections between active elements.

We have determined that the cellular automaton (CA) best satisfies the requirements for a successful post-microelectronics chip architecture. As

shown in Figure 1, a cellular automaton is a collection of simple, active devices which interact locally in discrete space and time. Each active device, or cell, is placed at one of the vertices of a regular lattice, as shown. The logic operations carried out by each cell are determined by a rule of interaction which could be as simple as a lookup table. This architecture has the important advantages of both structural regularity and not requiring long-range connections to perform complex mathematical operations.

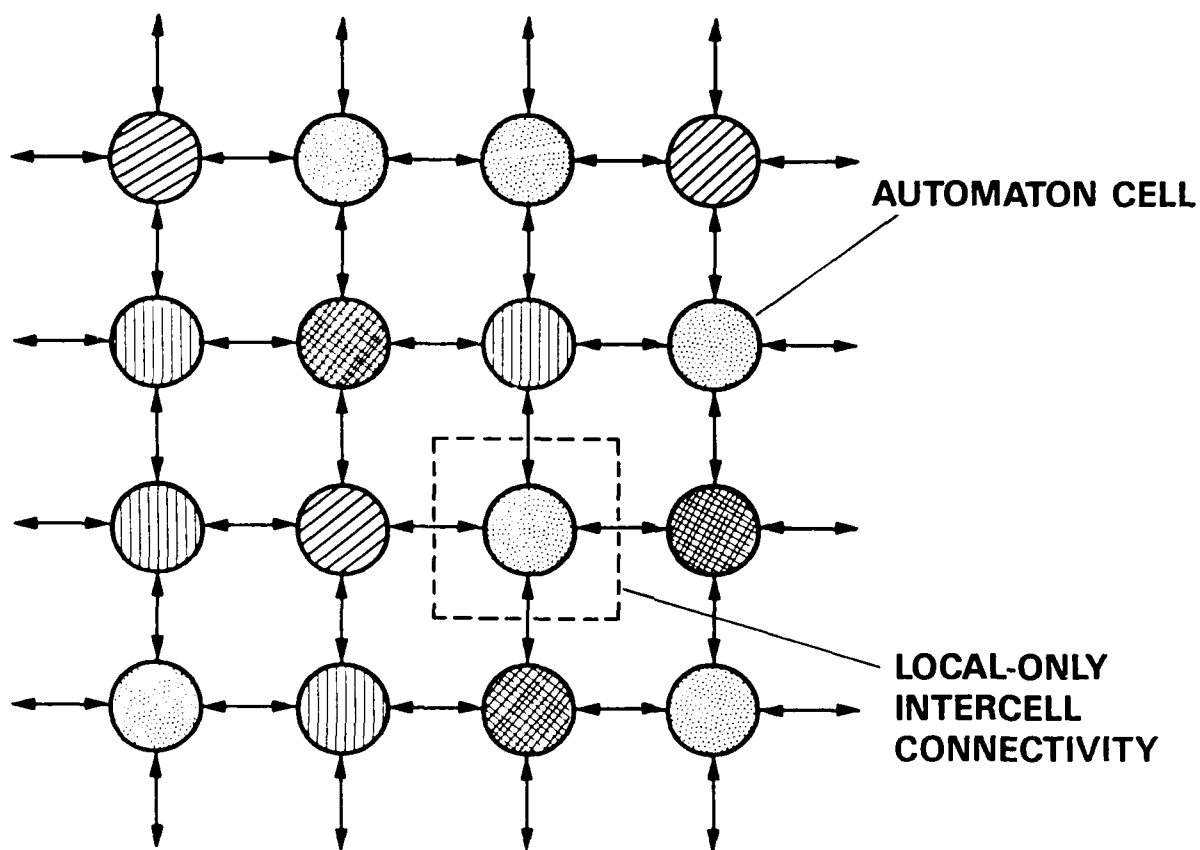


Figure 1. Two-dimensional lattice of a typical cellular automaton. Individual cells perform simple logic based on input from their immediate neighbor cells.

2. Objective of Architecture Research

The objective of this research is to provide computer architectures that avoid the performance barriers inherent in present computer technology. In particular, this work investigated device architectures that may permit the continued downscaling of integrated circuits through the mid-1990s and beyond. Ultimately, this goal will be achieved through the development of sophisticated and scalable computing functions that support next-generation, quantum coupled device technology. The near-term objectives of this effort were to quantify the dynamic behavior of general cellular automata and determine the unique physical properties of nanometer-sized devices that were useful for computation.

3. Research Approach

Computer simulations of multidimensional cellular automata were used to gain understanding of the dynamics and computational properties of arrays of locally coupled active devices. Large arrays of devices were characterized to quantify the sensitivity of locally coupled device architectures to noise. Finite automata theory was used to develop a multivalued algebra compatible with quantum coupled device technology.

Our strategy for developing quantum device architectures is shown in Figure 2. The physical properties of nanometer-sized electronic devices are expected to be quite different from those of conventional transistor-like switching devices. At present, there is no technology base with which to develop switch equivalents, logic gates, and complex functions. For this reason, we approached the investigation of quantum architectures from the bottom up. That is, we studied abstract cellular automata to determine empirically the behavior of large classes of locally coupled device arrays, and developed mathematical logic operations based on the most basic interactions expected from physical structures dominated by quantum mechanical effects.

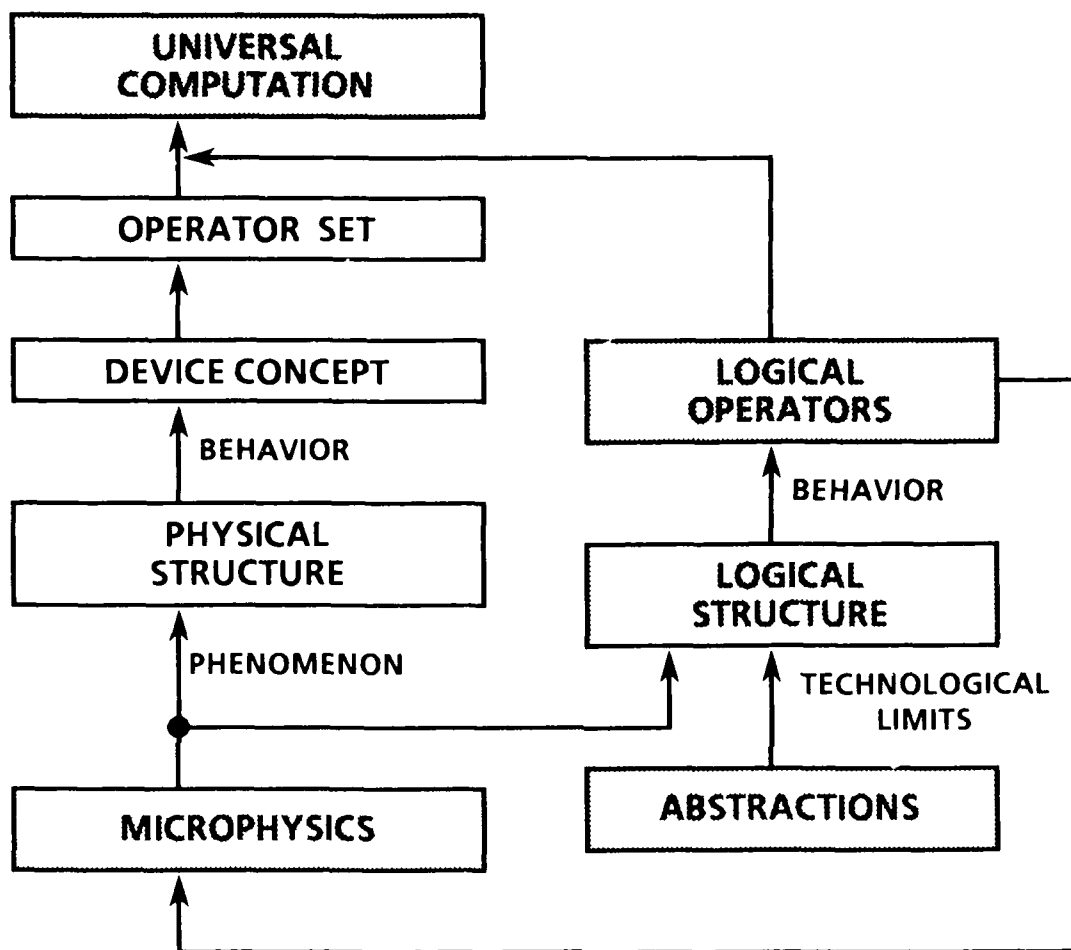


Figure 2. Nanoelectronics development path. Architecture path is highlighted.

4. Summary of Contract Results

4.1 Development of Simulation Environment

To accelerate the survey of the generic properties of locally coupled systems, three special-purpose, high-speed simulators were constructed. These workstation adjuncts were used to conduct detailed characterizations of the space-time dynamics of general cellular automata models. Each machine was based on either pipelined or parallel architectures. Processing throughputs as high as 1.2 billion cell operations per second were demonstrated with these machines. This throughput represented a 10,000X improvement over our software simulations using personal computers. We estimate that the data collected using these accelerators would have required over 3000 years of continuous simulation using a personal computer.

In addition to the hardware, a flexible software package was developed that allowed slower, but much more interactive, development of CA models. The combination of a user-friendly software environment with high bandwidth accelerators provided an excellent simulation environment with which to explore new concepts in computing CA, and then conduct exhaustive characterization of interesting models.

4.2 Characterization of One-Dimensional Automata

An understanding of the noise sensitivity of cellular automata may lead to methods for constructing fault tolerant computer functions. CA are characterized by dynamic attractors or Limit Cycles. These space-time patterns are the stable modes of a given distribution of rules and cell states. Knowledge of these eigenmodes fully describes the noise-free operation of general CA. In addition, the response of a model to added noise also quantifies the sensitivity of a particular model to event upset.

To obtain the statistical properties of one-dimensional CA, we used computer simulations to measure the Markov properties of a broad class of CA. It is possible to consider CA as multistate memories if we take the long-term evolution (attractors) as the possible items that can be recovered from the

memory. Therefore, it is useful to determine how noise affects the isolation between stable cellular automata behaviors. To this end, we measured the noise sensitivity of a large class of CA to added noise. First, nearest-neighbor connected cellular automata were characterized to determine the limiting time evolution of these models. The number and size of state attractors were determined as a function of rule and lattice size for all 256 interaction rules. To examine the noise sensitivity of these systems, Markov transition matrices were then tabulated based on Monte Carlo experiments on all rules and for all array sizes up to 28 cells in length. It was found that nearest-neighbor automata may be grouped according to the size and number of their state attractors, and that, in general, the long-term dynamics of these systems are not fault-tolerant.

4.3 Characterization of Two-Dimensional, Totalistic Automata

By analogy with majority logic approaches to error correction in noisy circuits, we examined the sensitivity of two-dimensional, totalistic cellular automata to noise. Under totalistic rules, the interaction between cells is a function only of the average state of their neighbor cells. As a result, a fraction of the cells within a totalistic array may be upset by thermal fluctuations, for example, without adversely impacting the overall dynamics. We found that, typically, the average state of the array cells could be controlled (switched) by varying the state of the perimeter cells. The model displayed transistor-like switch action, demonstrating gain and inherent resistance to multiple event upset. In addition, we were able to program hysteresis in the relationship between average cell state density, array size, and boundary conditions. The behavior of this class of automata is characterized by significant fault tolerance of both soft (recoverable) and hard (fixed) errors in the cell states. Due to the totalizing action of the cell rules, as many as 10% of the array cells could be defective without affecting qualitatively the switching behavior of the overall network. This response has important application in embedding fault management in the most basic logic operations of nanometer-sized circuits.

We also examined the ability of locally coupled arrays to store reliably randomly selected cell patterns. We demonstrated that several stable cell patterns could be stored, peripherally addressed, and retrieved from the same

totalistic array using simple correlation techniques. Patterns could be recalled by addressing the perimeter of the array with a subset of the desired cell pattern. These automata demonstrated a higher level of functionality by acting as fault-tolerant, associative memories. A method for enhancing memory recall through simulated thermal annealing was demonstrated. Experiments were conducted to determine the effective temperature at which a given model can no longer use totalistic averaging as a practical method for filtering noise. These data are not included here, but are currently being prepared for publication.

4.4 Development of Quantum Algebra and Circuit Concepts

Avoiding the limits of current VLSI technology will require the development of sophisticated device functions that transcend simple "transistor-like" switch operations. One approach uses multivalued logic concepts to permit higher radix operations between, for example, locally coupled quantum devices. Traditional multivalued algebras were not immediately realizable by the more obvious quantum interactions of resonant tunneling and quantum size effects. However, we showed in principle that algebras exist that are naturally compatible with quantum device properties. Quantum device structures were discovered that meet the basic mathematical requirements for a functionally complete logic system.

4.5 Self-Timed Cellular Automata

The physical scaledown of electronic circuits exacerbates many design problems associated with low-level connectivity and timing. Of particular concern is the problem of clock skew within and between functions. As circuit complexity increases, some form of mutual signaling must be used to synchronize independent functions. These clock signals provide an orderly method for data exchange between asynchronous functions and also supply the "arrow of time" needed to direct the flow of data within functions. However, the low dimensionality of the CA places obvious limits on the degree to which the activities of large device and function arrays may be synchronized. It is essential to devise generic methods for distributing skew-free clocks or to provide self-timing techniques compatible with nanometer-sized circuitry.

In studying asynchronous cellular automata, we devised a simple method for managing the timing of computation in cellular automata. A simple,

three-phase clock was implemented within each automaton cell. Under the control of these clocks, interactions between asynchronous cells could be effectively synchronized. The approach is most easily applied to nearest-neighbor coupled cellular automata, although any level of connectivity can be supported at the expense of cell complexity and computational throughput. We emphasized the use of the method for nearest-neighbor CA, since this class of automata is known to be functionally complete.

II. MOTIVATIONS AND OBJECTIVES OF ARCHITECTURE RESEARCH

This section presents a review of some of the problems with current chip technology and discusses the essential features required of a successful, post-microelectronic approach. While instructive, it may be skipped without loss of continuity.

1. Introduction to Problem

The information economy is a direct offspring of semiconductor technology. In large part because of the invention and improvement of the integrated circuit (IC), information management has become decentralized, convenient, and cost-effective. The further development of IC technology is fueled by an ever-increasing demand for information processing in science, medicine, education, industry, and national defense. Semiconductor integrated circuits are the primary enablers of defense electronic systems. This pervasiveness has continued due to the exponential improvements in the power, size, and weight factors of integrated systems with time. Certainly, the motivations to improve the performance of computing systems will intensify as we move into the age of widespread robotics, desktop supercomputers, man-machine interfaces, and hardwired artificial intelligence.¹

There are two approaches to providing the computational resources that will meet the information processing requirements of the 1990s and beyond. First, new algorithms can be developed that give exponential improvements in the mapping of a problem solution onto present technology. The Fast Fourier Transform is an archetypical example of a performance breakthrough achieved by algorithmic optimization alone. Embodiments of optimized algorithms range from systolic digital signal processor arrays to silicon neural networks. Second, processor performance can be enhanced by increasing the level of chip integration and functional density. Function scaledown, which also provides increases in speed, as well as reductions in power and weight, has been the dominant system enabler to date. In fact, the growth of the semiconductor industry has paralleled the physical downscaling of computer components.

Until recently, the potential for advancing both algorithm science and chip technology was open-ended. There are now clear indications that chip-level functional density will saturate within 20 years. When this maturation of technology occurs, the return on the investment in new algorithms will also decline. The resulting slowdown of chip enhancement will impact the full spectrum of semiconductor computer technology. The impending catastrophe cannot be avoided by anything short of a revolution in chip technology.²

2. The Scaling Limits of Conventional Technology

An important factor responsible for the pervasiveness of the integrated circuit has been the sustained exponential decrease over time of minimum lateral circuit geometries. However, there are limits to the use of geometry scaledown as the means to further improve the performance of conventional integrated circuits. These limits are so fundamental as to force the ultimate abandonment of high connectivity architectures, transistors, and the classical circuit concept. We can review briefly a few of the basic problems.

2.1 Transistor Scaling Limits

The linchpin of solid state electronics is the p-n junction. The depletion layers between p- and n-regions of a circuit serve as the potential barriers required to guide the flow of electronic charge. A transistor is simply a device that can modulate the effectiveness with which p-n structures electrically isolate two points in a circuit. Any integrated technology must possess a similar ability to control the direction and magnitude of charge transport. Unfortunately, depletion layers begin to lose their ability to confine electrons as p-n widths are scaled below 0.2 μm in size. Based in part upon the failure of depletion isolation, it has been estimated that junction-based switching devices cannot be shrunk appreciably below 0.1 μm .³ The classical transistor must eventually lose its ubiquity.

2.2 Connectivity Scaling Limits

The overall functional density (functions/area-time) of a computing machine is controlled by the available interdevice communication bandwidth and device density at the board, chip, function, and gate levels. Since the speed of light sets an upper limit on communication rate, system performance

can be improved if more functions are integrated on-chip. However, current models of circuit parasitics show that with scaling, RC delays and interline coupling rapidly degrade the speed and noise performance of submicrometer interconnections.^{4,5} Moreover, device size and electromigration factors set upper limits on the available current density at the device port. For this reason, even well-isolated, long interconnects will have self-capacitances that lead to unacceptable effective device switching speeds. In the near future, the computational advantage of long, multilevel interconnects will be negated by the saturation and/or reduction of the effective communication bandwidth per wire. Therefore, merely combining submicrometer active devices with conventional interconnect networks will not significantly improve the performance of integrated systems.

2.3 Yield Scaling Limits

At present, the unavoidable errors induced by substrate defects, cosmic rays, and thermal fluctuations are second-order considerations in VLSI design. A low density of fixed and transient errors can be handled by production culling and error control coding, respectively. Further component scaling will make these ad hoc fault management schemes obsolete. Scaledown reduces the number of electrons that can participate in each computation. This reduction translates into an increase in both the informational impedance and the noise sensitivity of the switching event. In the future, media noise and transient upset events will affect entire groups of submicrometer devices. Hard and soft faults will become inherent technological characteristics of ultra-scaled structures. The age of the 100% functional, fault-free integrated circuit is fading fast.

2.4 Addressability Scaling Limits

An inherent problem with ultra-integration will be the further decrease in our ability to access directly a given functional resource. For a minimum feature size ($1/S$), resources may grow as fast as S^2 and S^3 for two- and three-dimensional circuits, respectively. However, conventional access methods are essentially peripheral, so that I/O accessibility may always be one dimension behind component density. With scaledown, it will become increasingly difficult to address directly either single devices or even

entire device groups. This growing inaccessibility presents the potential for severe I/O bottlenecking at function boundaries.

2.5 Superposition Limits

The issue of intercomponent crosstalk points out a subtle influence scaling will have on the circuit paradigm. Traditionally, a Principle of Superposition has been assumed in the design of complex information processing structures. This principle, quite valid in the 1970s, asserted that the properties of active elements such as transistors and logic gates were independent of the physical environment of the device. In addition, the superposition argument maintained that the behavior of the computing system at any level of complexity could be expressed as a piecewise-linear sum of the characteristics of more basic functions. Many of the impediments to the further enhancement of conventional computing structures can be traced to the invalidation of superposition at one or more levels of complexity. For example, it is clear that successfully avoiding the problems associated with interconnect scaling would allow device density to control the limiting on-chip functional density. However, as interdevice geometries decrease, the coupling between these active, nonlinear agents will also increase. Device densities are now reaching the point where classical notions of isolated, functionally independent active elements are less appropriate than are distributed models that include the possibility of collective modes of device interaction. Conventional device simulation models already recognize the need to account for interdevice parasitics.⁶ Scaling into the nanometer size regime only exacerbates the problem of isolating device function from the local environment. Further scaling will make it impossible to partition a circuit into active and passive components. Lumped-constant rules will be of little use in nanometer design.

In addition to architectural problems at the device level, certain chip applications cannot employ the principle of superposition in the design of functions. Even though traditional sequential computers are computationally universal, the complexity of NP-complete optimization problems requires prohibitively large serial processing power to reach good solutions within an acceptable time. Important tasks such as artificial intelligence, image understanding, adaptive learning, and general speech recognition demand

massively parallel, global computations. It is possible that true machine intelligence can evolve only if the behavior of the system as a whole exceeds the sum of its separate parts.⁷ If that is true, even highly concurrent architectures that can be decomposed into modules of more primitive functionality may not exhibit the emergent collective properties required to solve important compute-bound problems.

2.6 Limits to Static Design

Interpretation and compilation are popular approaches to hardware and software development. The two methods trade speed for flexibility. In program compilation, the computer programmer expresses an algorithm in terms of a language, then converts this logical literature into a fixed, machine-dependent set of basic operations. Similarly, the computer architect may employ a library of hardware primitives that can be used with a "silicon" compiler to map algorithms into an application-specific integrated circuit (ASIC). In contrast, program interpretation uses a core routine to execute arbitrary algorithms phrase by phrase, while machine interpretation utilizes a microprocessor to emulate arbitrary hardware at reduced throughput.

These methods suggest that it is not possible to have, simultaneously, algorithmic flexibility and optimum throughput. The more optimized the hardware, the lower its adaptability to new computing environments. For this reason, ASIC may be destined to accelerate, but never displace, general-purpose, possibly dinosauric architectures. Our inability to "erase" transistors and wires as easily as we erase code warns of an ultimate saturation of the scope-performance product of static architectures.

3. Essentials For a Successful Post-VLSI Era

The aforementioned limits are compelling, but are for the most part technological in origin. There is no fundamental reason why physical scaledown cannot be used to improve the functional density of integrated circuits indefinitely. However, dramatic shifts from convention are necessary if the advantages of scaling are to be realized beyond the 1990s. To avoid the limits of current practice, we have proposed the following set of six essential requirements for a successful postmicroelectronic IC technology.⁷

- Devices must be scalable to fundamental physical limits.
- The most basic functions of a computing system must employ only local connection schemes.
- Device function must transcend simple switch operations.
- Intercomponent coupling must be exploited as a method of communication and control.
- Fault management must be incorporated at the lower levels of functionality and feature size.
- Functionality must be reconfigurable.

3.1 Discussion

Further increases in the density of on-chip computational resources can be obtained only through the combined development of revolutionary devices that are based on nanometer physical phenomena and chip architectures that avoid interconnect saturation. This implies an absolute minimization of device connectivity within low-level functions. Therefore, it will be necessary to recast existing random logic networks into low-connectivity equivalents. To avoid I/O bottlenecks in the reorganized functions, a high degree of functional concurrency and pipelining will be necessary. We expect that the advantages that accrue from regular VLSI layout will carry over to regularized nanometer functions. The combination of layout regularity and pipelining at several levels of complexity suggests that these architectures will have a measure of topological scale invariance. In short, the future of chip architectures may be very fractal.

It will be necessary to construct functions holistically through an appreciation of the impact of environment upon active and passive structures. Integrated circuit models in the late 1990s may abandon entirely the idea of point-in-space design in favor of the construction and solution of the continuous partial differential equation that describes the effective computational volume.

To compensate for the penalties incurred by reducing interconnect complexity, device behavior must transcend simple "transistor-like" functions. By "device" we mean a set of physical structures that collectively perform a computing function without the use of wiring. Since reduced connectivity translates into communication delays, these most basic computing elements must execute higher level operations when the data finally arrive. Increasing device complexity amounts to extracting greater functionality from known physical phenomena. For example, devices based on superconductivity effects demonstrate how function can be wrought from phenomena. Submicrometer Josephson Junction (JJ) technology has been used to construct an A/D convertor using only one JJ device per digitized bit.⁸ This significant improvement in functional density arises not from the cleverness of the device architecture, but directly from the behavior of very small, very cold things. A similar strategy must be used to leverage nanometer physics into complex device function.

The negative effect of scaledown on signal-to-noise ratios emphasizes the need for a generic method for managing soft and hard failures within ultra-scaled circuits. Current efforts to develop fault-tolerant architectures at the system and processor levels must be extended to include fault management at the most basic levels of functionality. Finally, to route around defects, increase adaptability, and avoid early obsolescence, chip functions must be reconfigurable. This capability will also eliminate many design turnaround problems by softening the distinction between software and hardware.

In summary, successful next-generation chip architectures must be based on minimal connectivity and sophisticated active devices. Future chip functions must be pliable and fault-tolerant. In the next section we discuss candidate devices and architectures for meeting these criteria.

III. NANO-ELECTRONICS

The complete approach to meeting the postmicroelectronic challenge of next-generation IC technology is called Nanoelectronics.⁹ (The more descriptive term, nanoelectronics, has become popularized over the original term "ultrasmall electronics.") This section reviews the basic concepts of nanoelectronic devices and architectures.

1. Nanoelectronics Technology

1.1 Quantum Coupled Devices

Conventional electronic devices are designed from a set of approximations that allow explicit quantum mechanical considerations to be ignored. However, as dimensions scale below $0.1\ \mu\text{m}$, new effects associated with the wavelike nature of the electron begin to dominate the mechanisms of charge transport. Recent advances in materials science have made it possible to construct devices that demonstrate strong quantum effects. These non-classical phenomena, including quantum size effects and resonant tunneling, can be used to build a new technology base for nanometer electronics.

Semiconductor technology now provides the means to construct electronic nanostructures. Heterostructure techniques combine dissimilar semiconductors layer-by-layer to tailor the effective electronic band structure of the resulting composite on an atomic scale. The energies and densities of electron states can be engineered into a heterostructure by controlling the stoichiometry of the final chemical compound. Popular fabrication methods include molecular beam epitaxy and metal-organic chemical vapor deposition.¹⁰ As a heterostructure is grown, abrupt changes in composition can be used to shift electronic material properties. Unlike broad, fuzzy p-n junctions, the effective potential variations across a heterostructure are sharply defined.

Although the required processing technology is still under development at Texas Instruments and elsewhere, we will take nanofabrication as a given and consider a nanoelectronic structure that incorporates most of the relevant physical phenomena required to build a useful nanometer-sized device. These physical properties will form the basis of compatible logic and architecture systems. We consider two nanometer-sized cubes of semiconductor that are embedded in a sea of another material as shown in

Figure 3. For clarity, we will use the aluminum gallium arsenide (AlGaAs) material system in our discussion. The energy diagram for the possible electron energies in this structure is shown in Figure 4. Due to the difference in the electronic properties of GaAs and AlGaAs, the conduction electrons within the GaAs material are surrounded by an electrostatic potential barrier. In addition, the small size of each cube squeezes out most of the normal energy states of conduction band GaAs, leaving only a few sharply defined energy levels, as shown in the potential well diagrams. State quantization due to dimensional scaling is known as the quantum size effect (QSE).¹¹ To first order, the number of distinct electron energy levels within a potential well is inversely proportional to the square of the well width. Using the dimensions of Figures 3 and 4, our hypothetical GaAs structures would have only a few bound states, as shown. The QSE can have a controlling influence on device properties even if only one or two physical dimensions are quantized.¹⁰ As we will see below, the exploitation of the QSE is basic to nanoelectronics. Physical structures that display size quantization effects are called quantum wells. Three-dimensionally quantized structures, like our quantum cubes above, are called quantum dots.¹²

Classically, electrons could be trapped forever inside potential cages. However, the wave properties of electrons enable them to escape from quantum wells by the process of resonant tunneling.¹¹

The resonant nature of electronic tunneling between quantum wells also provides the mechanism with which to control charge transport. Charge can be exchanged between quantum wells if:

- (1) They are in close proximity so that tunneling effects are strong,
- (2) Energy is conserved,
- (3) Momentum is conserved, and
- (4) The destination state of the transported charge is unoccupied.

These conditions are satisfied when GaAs-AlGaAs quantum wells are spaced by less than a few hundred angstroms, and a compatible occupied state in one

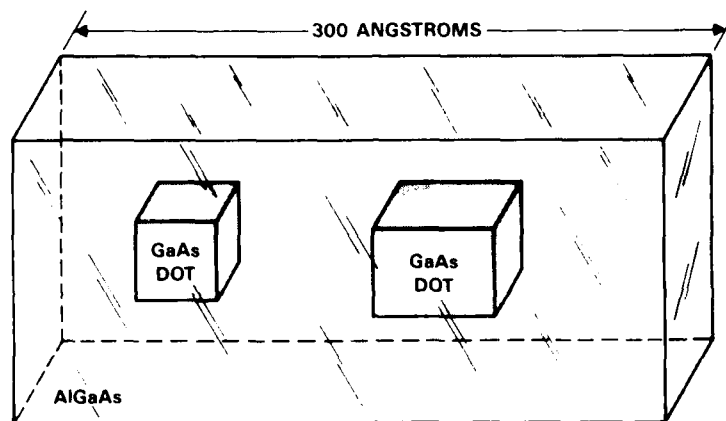


Figure 3. GaAs quantum dots embedded in a sea of AlGaAs.

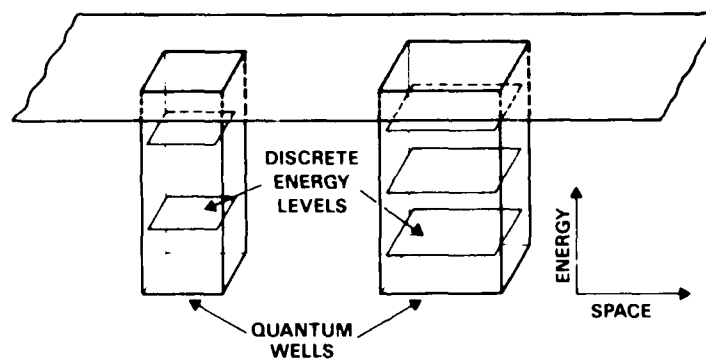


Figure 4. Potential energy diagram for conduction bands of quantum dots.

well is in energy resonance with an empty state in the other. A schematic of the charge transfer process between two quantum dots is shown in Figure 5. As shown in Figure 5, energy is easy to conserve in tunneling between identical quantum wells, since symmetry requires the allowed energies in both wells to be the same. The tunneling probability between dissimilar or biased wells is determined by the degree of alignment or resonance between energy levels. In principle, the communication between dissonant quantum wells can be reduced to an arbitrarily small value.

The quantum size effect and resonant tunneling provide all the essential phenomena required to control the direction and amplitude of charge transport in nanostructures. The QSE forces bound charge to take on discrete energy values in the form of quantum levels. RT is then used to exploit this quantization as a means to confine or direct charge flow between quantum wells. Devices that operate ostensibly through the use of the QSE and resonant tunneling effects are called quantum coupled devices. Preliminary ideas for devices based upon resonant tunneling have been conceived.⁹ In addition, the empirical and theoretical understanding of transport in quantum devices has increased considerably in recent years.^{12,13}

1.2 Cellular Automata Architectures

Effective utilization of the scaling advantages of quantum coupled devices requires the development of equally scalable device architectures. This requirement is best satisfied by the cellular automaton.¹⁴ Basically, a cellular automaton (CA) is a collection of simple active devices that interact in discrete space and time. Each active device, or cell, is placed at one of the vertices of a regular lattice as shown in Figure 1. The dynamics of a CA is controlled by the independent, but highly parallel, interaction of each cell with its local environment. The function of each cell is specified by a rule of interaction analogous to a state transition lookup table. Cell rules are usually deterministic functions of the instantaneous value of the states of nearest-neighbor cells. However, cell types may be defined that allow direct interaction with more distant cells and that include long-term memory. More elaborate models can test the impact of clock skew and fault tolerance by allowing cells to interact asynchronously and obey time-dependent rules. Spatio-temporal "snapshots" of

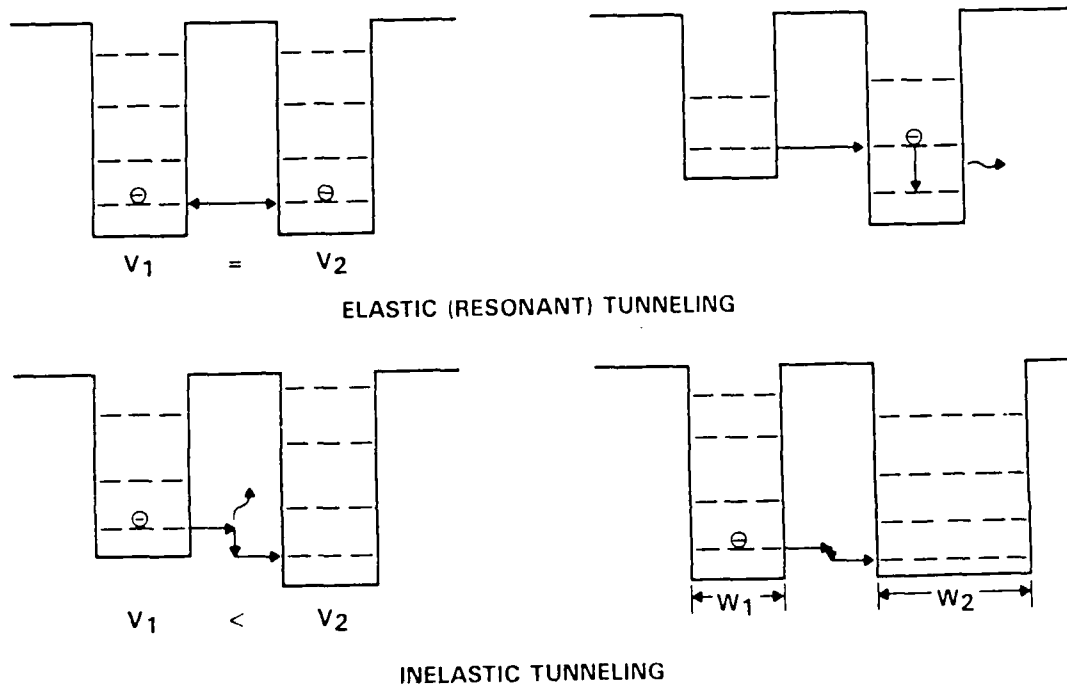


Figure 5. Elastic (strong) and inelastic (weak) tunneling between quantum wells.

the dynamics of several one-dimensional cellular automata are shown in Figure 6. All these examples demonstrate nearest-neighbor interactions between two-state cells. Light areas (paper) represent state "0" cells, while dark areas (print) correspond to state "1" cells. This simple class of CA displays a variety of behaviors ranging from time-independent to nearly chaotic time evolution.

The most basic characteristic of a CA is the formal limit on the range of direct coupling between lattice cells. Typically, cells are only allowed to influence the dynamics of nearest and perhaps next-nearest neighbors. Thus, CA embody the precepts of eliminating long interconnects while also explicitly defining the behavior of a cell in terms of a tightly coupled local environment.

Computation in a CA results from the adjustments each cell makes to its internal state in response to changes in the state of its local environment. During a computation, all cells act in parallel, using their initial state and individual interaction rules to calculate future lattice states. Despite minimal connectivity, many CA are known that can perform all the general logic operations required to build a computer. The Game of Life is a popular example of a two-dimensional CA that can be set up to perform general-purpose computation.¹⁵

1.3 Combining Quantum Coupled Devices and Cellular Automata

The close analogy between quantum coupled device arrays and cellular automata can be seen by considering the one-to-one mapping of quantum devices with CA lattice cells. An automaton cell is completely defined by its interaction rule, which determines how information flows through the network. In an analogous fashion, the electrostatic coupling between quantum wells provides a mechanism for tailoring charge flow between wells. Conditions of resonance and dissonance between quantum structures are equivalent to connections and isolations between cells, respectively. As shown in Figure 7, the strength of interwell coupling can be modulated strongly by the electric fields that are broadcast from each well to its neighbors. Lithographic tuning of quantum well energy levels can be used to express the variations in cell type and connectivity. We call the combination of quantum devices and CA a quantum cellular automaton.

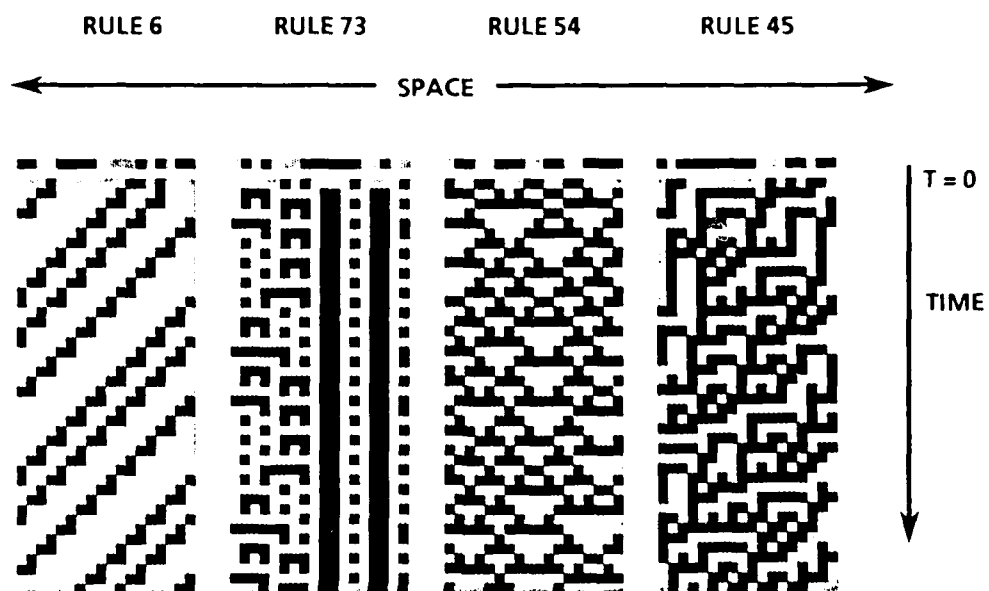


Figure 6. Time evolution of cellular automata.

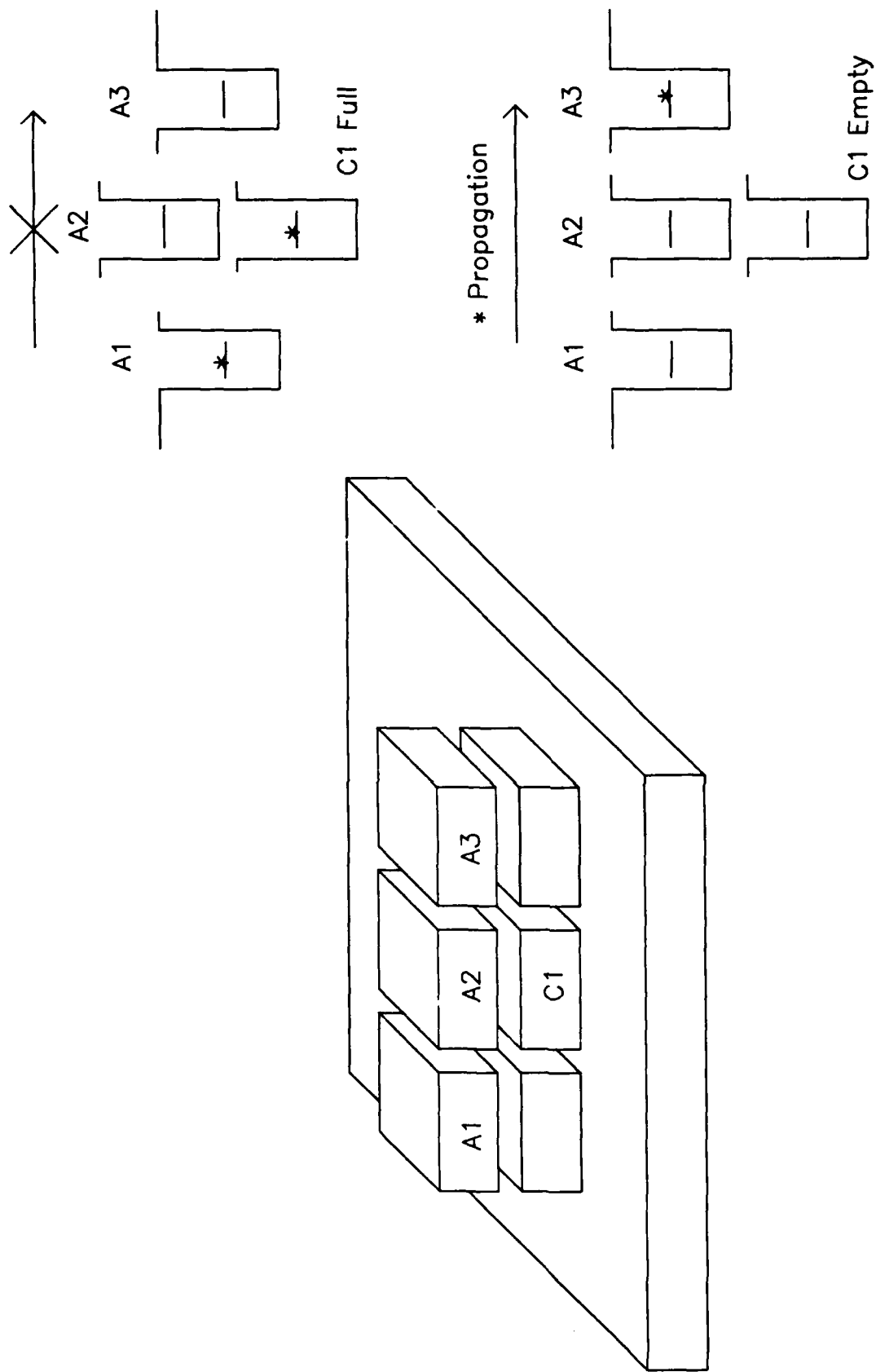


Figure 7. Interwell effects can gate charge transport. Assume that well C1 is electrostatically coupled to well A1, but not RT-coupled.

IV. SUMMARY OF SIMULATION TOOLS DEVELOPED

Special software and hardware simulation tools were developed to support the study of one- and two-dimensional cellular automata. Software was used to explore new cellular automata concepts while high-speed hardware was used to accumulate detailed statistical data on particular model systems. The combination of interactive software with high bandwidth accelerators provided an excellent workstation environment for this research.

1. Software Simulation Tools Developed

A Texas Instruments Professional Computer (TIPC) was used to support the software simulations conducted during this program. To maximize simulation throughput, many of the basic simulation/analysis routines were written in assembly language. Pascal was used to link routines and manage the real-time user interface. We found this software combination ideal for rapid testing of ideas as well as for supporting longer-term background simulations. A hardware random number generator was added to accelerate simulations involving asynchronous and stochastic cell interactions. The final software environment allowed the simulation of one-dimensional cellular arrays of up to 512 cells, and two-dimensional arrays of up to 128 by 128 cells. Simulation rates as high as 10,000 cell operations per second were obtained through software pipelining. The graphics software limited the visual display of large array simulations to a maximum of four array-updates/second.

In the typical simulation, abstract cells would be located on a square lattice and constrained to interact according to predefined rules. Each cell could be represented by either a combinatorial logic function or lookup table. The rule associated with a given cell was used, along with a knowledge of the state values of neighboring cells, to compute the future state of that cell.

To permit real-time investigations of time-dependent rules and inputs, function keys were defined which enabled a simulation to be paused, altered, and then continued. Programmable levels of noise could be added to individual cells, local cell groups, or the entire lattice. This latter facility was used when testing the sensitivity of array dynamics to random fluctuations in cell behavior. To assist in the spectral analysis of the

spatio-temporal behavior of each model, an option was provided for performing either the spatial Walsh transform of a group of cells, or the temporal Walsh transform of a single cell, in parallel with each simulation step.

2. Hardware Simulation Tools

The detailed characterization of a wide variety of cellular automata required greater simulation throughput than that available from software. As shown in Table 1, significant speedups relative to conventional computers are possible through the use of data pipelining and parallel approaches to cellular automata simulation. To this end, we designed and constructed several hardware simulators, to support the high-speed simulation of one- and two-dimensional cellular arrays. These machines were used to conduct detailed characterizations of several interesting CA models that were developed using the software workstation. With average throughputs as high as 1.2 billion cell operations per second, this hardware provided 100X to 10,000X performance speedups over software simulation alone. These speedup factors were found essential to accumulating meaningful statistics on the models studied during this program. We estimate that the data collected using these simulators would have required over 3000 years of continuous simulation on a single personal computer.

Table 1. Relative Performance of Cellular Automata Simulation Architectures

Architecture	Throughput	Normalized
Von Neumann	$N \cdot M \cdot R$	$N \cdot R$
Pipelined	$N \cdot M$	N
Systolic	$M \cdot R$	R
Full Parallel	M	1

Legend

N Number of cells in simulated array
M Convergence time
R Number of connections per cell

Block diagrams of the systems are shown in Figures 8, 9, and 10. The machines schematized in Figures 8 and 9 were based on pipelined architectures and included provisions to vary in real-time: (a) the interaction rules assigned to each array cell, (b) the synchronization between cell state transitions, and (c) the degree of random noise associated with each cell state. The simulator shown in Figure 10 operated in a fully parallel fashion, but was designed to simulate only fixed-rule, noiseless, one-dimensional cellular automata.

A basic tenet of our research concludes that useful nanoelectronic device architectures will be characterized by very short-range connectivity patterns. For this reason, we restricted intercell connections, as simulated by the hardware, to next-nearest-neighbor cells or closer. Under this restriction, all simulation models could be mapped into the cell arrays shown in Figure 11. The ends of each simulated array were connected to form a simple ring or torus as shown. The ring/torus structure provided periodic boundary conditions to each cell and minimized end effects during simulation of very large arrays. However, since our designs also allowed each cell to operate according to its own interaction rule, it was a simple matter to create constant-valued perimeter cells which then emulated fixed boundary conditions.

To maximize throughput, each simulator used lookup tables to hold data and control information. One set of tables served as a sequential state machine to control processing activities. Additional memory retained cell states and cell interaction rules. To update the state of an array of cells, small groups of cell states were bundled together to form a physical memory address. These addresses were then applied to the rule tables to compute in one clock cycle the next state of each cell group. Once a simulation model was microcoded into local memory, the machine was initialized with cell data and then allowed to operate independently under an internal clock for a fixed number of iteration steps before software control was returned. The partial autonomy of the hardware dramatically reduced the time overhead associated with data exchange to and from the host computer. The details of the simulator shown in Figure 10 are being prepared for publication in The Review of Scientific Instruments.

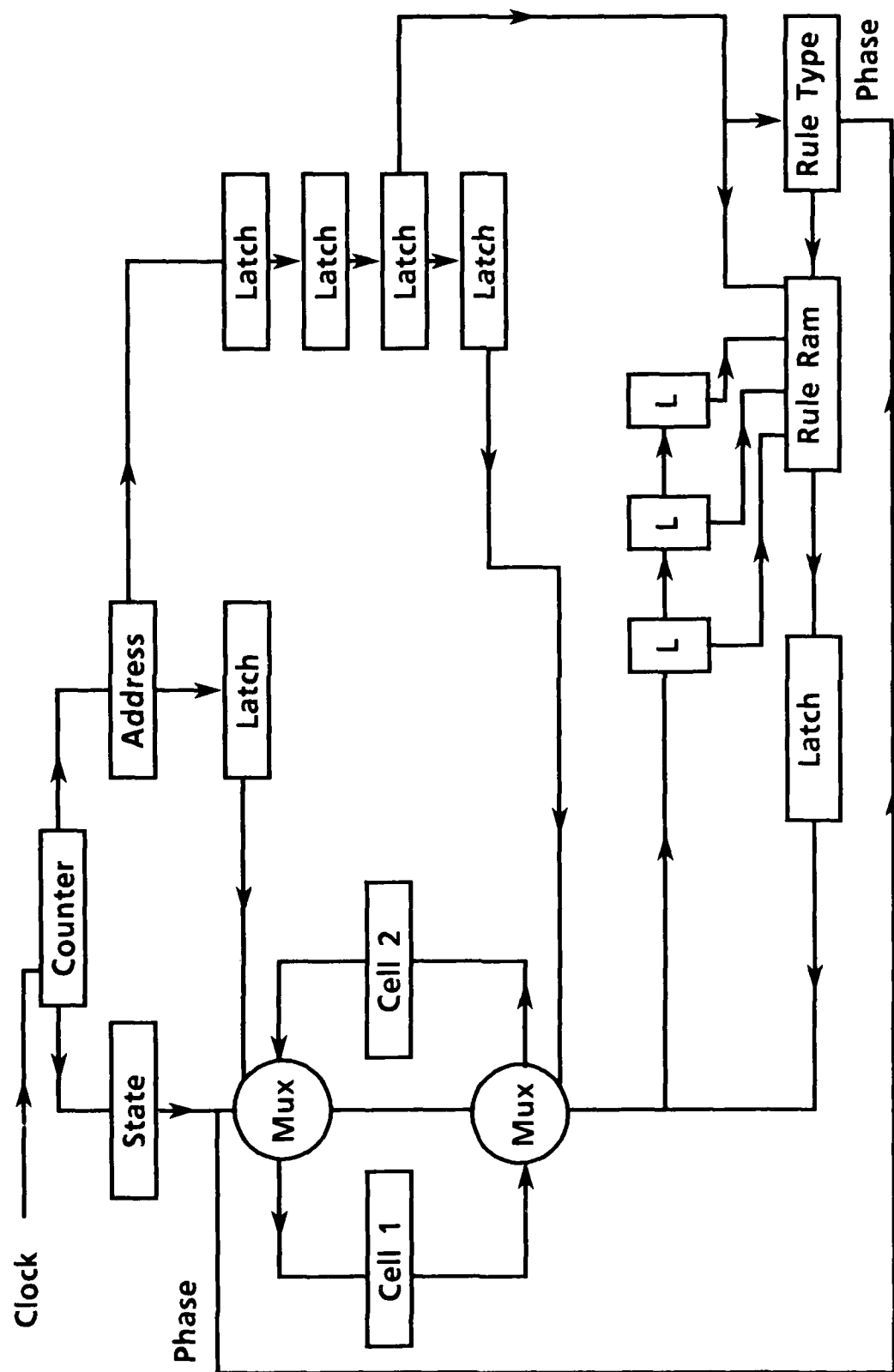


Figure 8. Flow diagram of pipelined one-dimensional cellular automata simulator.

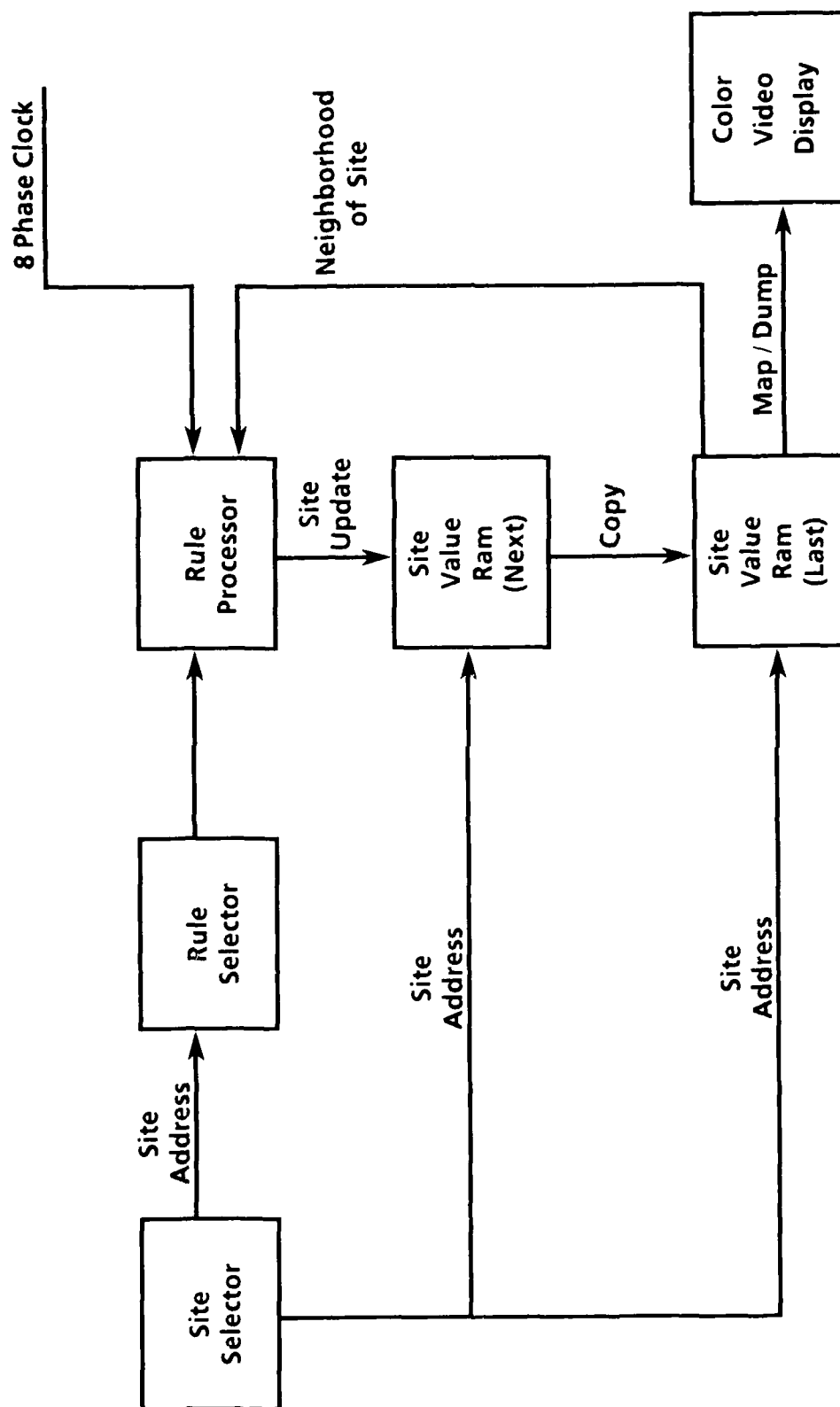


Figure 9. Flow diagram of high-speed, two-dimensional cellular automata simulator.

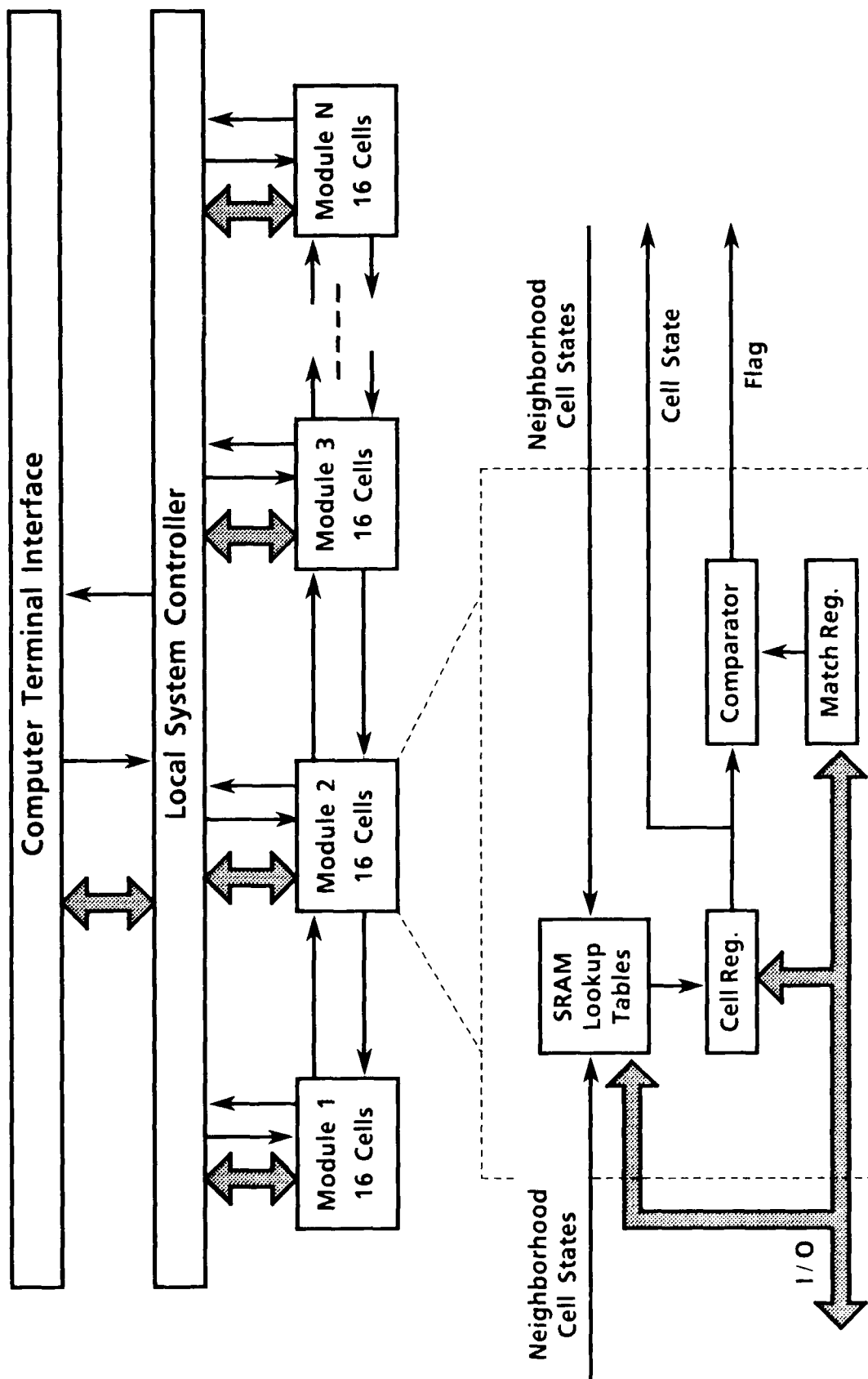


Figure 10. Flow diagram of fully parallel, one-dimensional cellular automata simulator.

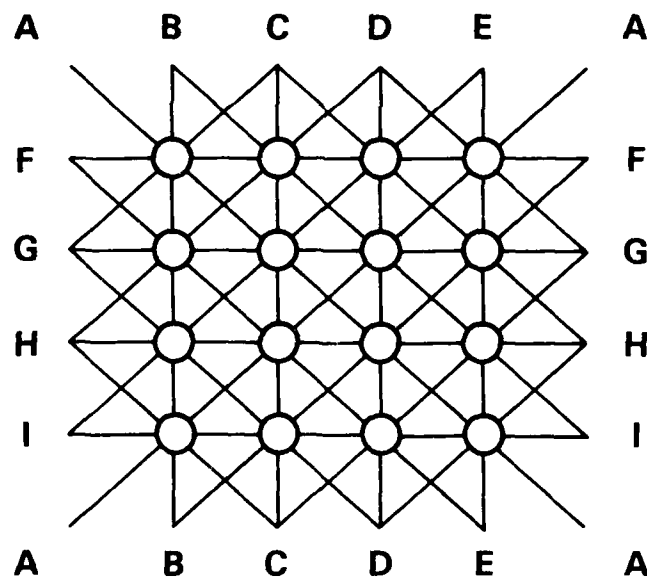
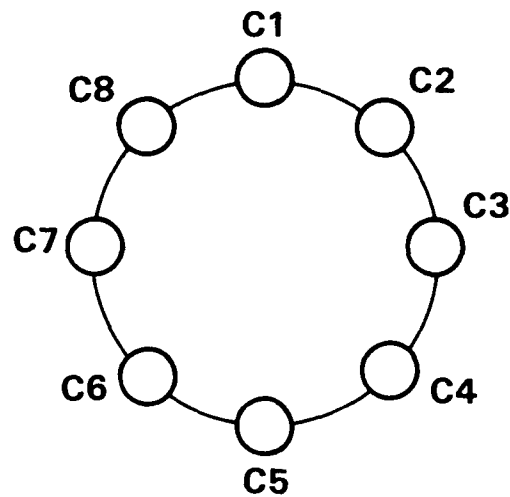


Figure 11. Ring and torus boundary conditions for next-nearest-neighbor cellular automata. Lines indicate direct intercell connections.

V. CHARACTERIZATION OF ONE-DIMENSIONAL CELLULAR AUTOMATA

Nearest-neighbor coupled cellular automata (CA) are characterized by local connectivity, simple cell interactions, and discrete dynamics in space and time. Despite the simplicity of this CA class, a wide variety of complex behaviors can be observed in the short- and long-term dynamics. An understanding of the structure of state bifurcations in noisy cellular automata may lead to methods for constructing fault-tolerant cellular automata computers. A noise analysis could be used to determine the effect of thermal fluctuations and cosmic rays on models that emulate an actual technology. Therefore, it is useful to characterize CA to determine their sensitivity to stochastic events.

As part of our research, nearest-neighbor connected cellular automata were characterized to determine the limiting time evolution of these models. The number and size of state attractors were determined as a function of rule type and lattice size for all 256 interaction rules.

To quantify the noise sensitivity of these systems, Markov transition matrices were tabulated based on Monte Carlo experiments on all rules and for all array sizes up to 28 cells in length. In accomplishing this objective, we determined the following properties of nearest-neighbor coupled, one-dimensional cellular automata:

- (1) The minimum set of rules that span the entire class of behaviors.
- (2) The periodicity of all stable points in the state space of each automaton rule.
- (3) The probability that a random initial cell pattern will result in a particular long-term evolution.
- (4) The resistance of all rule types to low rates of single event upset.

The quantitative results of these measurements are shown in Appendices A through F. These results are being prepared for publication.

1. Theoretical Development

1.1 Basic Dynamics of A Cellular Automaton

We can consider the instantaneous state of a cellular automaton to be represented by a state vector S , with dimensionality equal to the array's size. Starting from the current cell state configuration $S(t)$, the time evolution of any CA can be found from

$$S(t+1) = O_R[S(t)], \quad (1)$$

where O_R is a globally applied operator that carries out the iteration rule R assigned to the array. The recursive application of the rule operator can be written as O_R^N such that

$$S(t+N) = O_R^N[S(t)]. \quad (2)$$

For time-irreversible CA, all initial patterns $S(0)$ are mapped by the action of the rule operator into a set of periodic patterns called Attractors, or alternatively, Limit Cycles. This behavior is shown schematically in Figure 12. These repeat sequences are characterized by the equation,

$$S(t+L_i) = S(t) = O_R^{L_i}[S(t)]. \quad (3)$$

In the equation above, L_i is equal to the length of the relevant limit cycle.

We can further define a vector L , which is composed of the the distinct limit cycles associated with a given CA rule and array size. It is clear that for a deterministic CA, once entered, the dynamics cannot exit a limit cycle. That is,

$$\frac{dL}{dt} = 0. \quad (4)$$

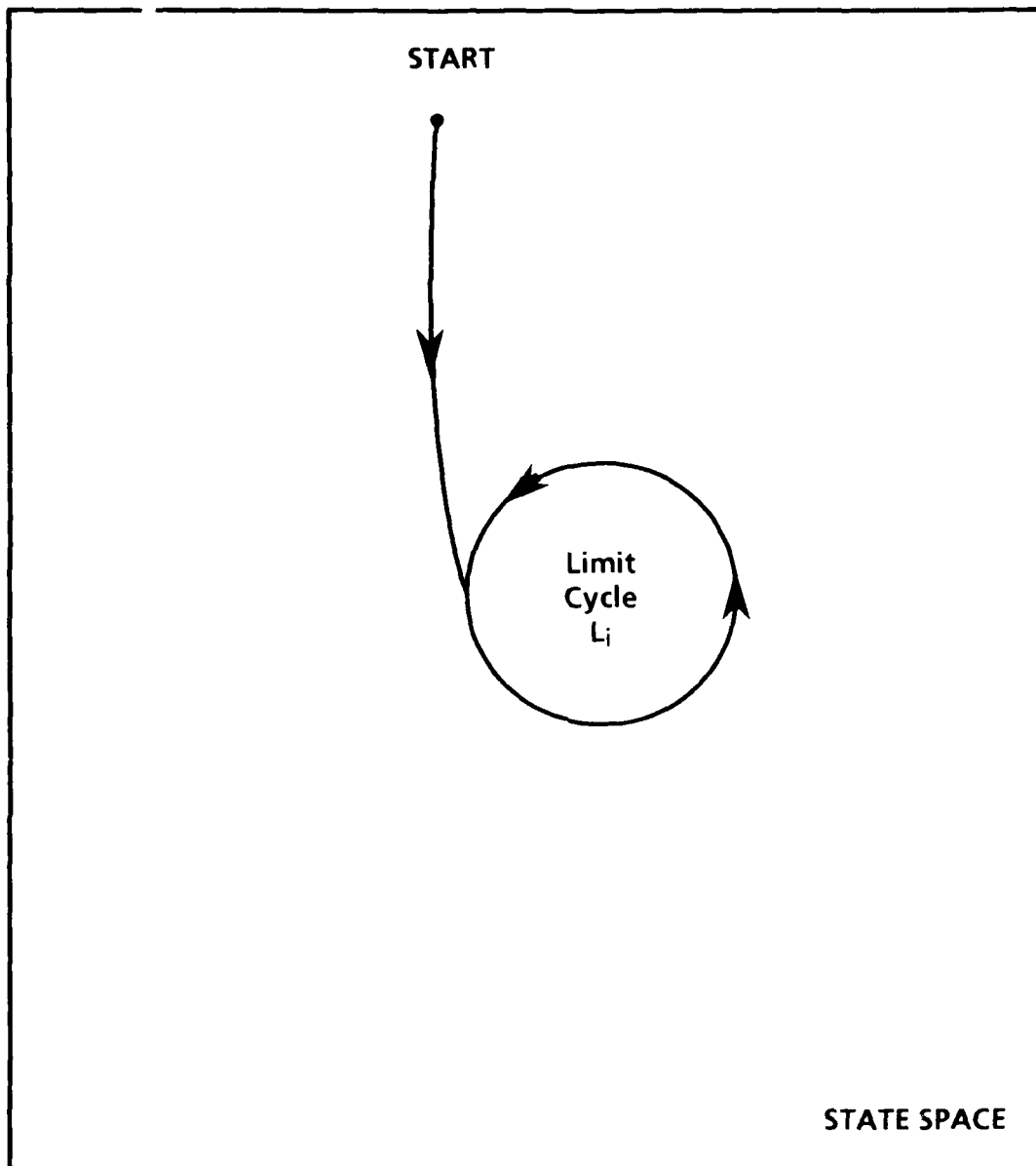


Figure 12. Schematic representation of limit cycle behavior for time irreversible CA.

In the presence of noise, the rule operator, O_R , must compete with stochastic events which destroy dynamic determinism. A CA composed of strongly attractive limit cycles will tend to resist minor fluctuations in state. A noisy system characterized by weak attractors will tend to hop among the possible limit cycle states in a more chaotic manner. We have summarized in Figure 13 some of the space-time events that can occur in dynamical systems. Deterministic CA are primarily Class C, with the behavior becoming increasingly Class A with increasing levels of cell-state noise. Attractor basins span essentially all state space in the general case. Class B dynamics such as singularities, which precisely conserve state space across a state transition, are rare. With reference to Figure 13, the resistance of a CA rule to noise is quantified as the fraction of Class A, B, and C events that make up the total state trajectory in the presence of noise.

Any CA can be made indeterministic by adding noise to the cell states. Random changes in cell state simulate physical events including thermal fluctuations and cosmic rays. A noise-sensitive CA has the potential to form the opposite of an attractor; a Repeller. It is usually not possible to predict the noise sensitivity of a CA network on the basis of an examination of the cell rules. In most cases, empirical determination of the fault-tolerance of each model is required.

An attractor maps two or more initial state patterns into a common future trajectory. If initial states are distinguishable only by noise, then an attractive CA can recover from noise events in a natural manner. In contrast, repellers formed by noise will be highly sensitive to noise if their basic indeterminacy is due to some stochastic process.

In contrast, the presence of noise in either the cell interaction rule or cell state can lead to the opposite effect: an ever-increasing bifurcation of state trajectories until, at some level of noise, the system becomes chaotic.

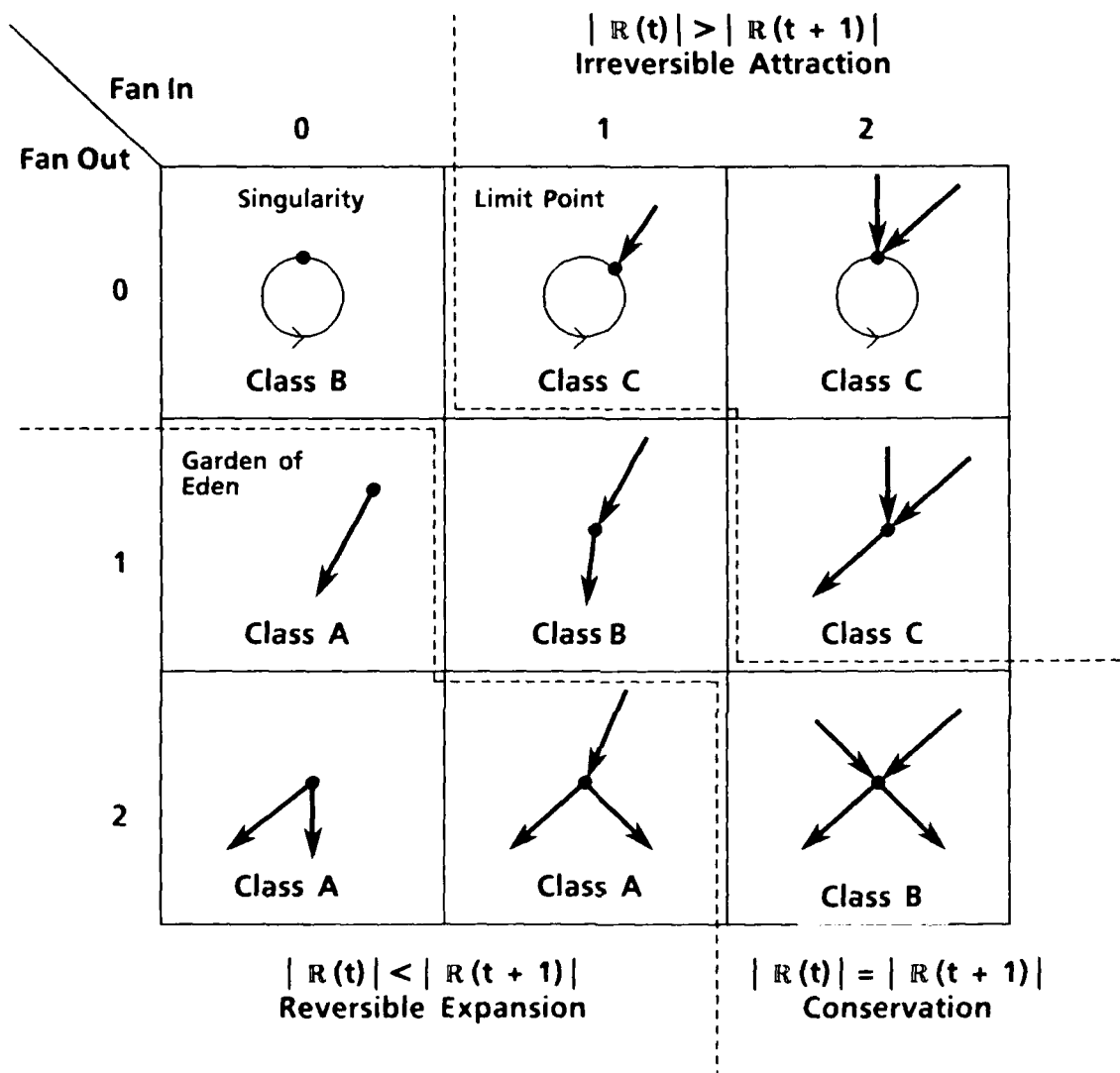


Figure 13. Trajectory classes in cellular automata. Class C behaviors are irreversible and deterministic, while Class A behaviors are indeterminate and reversible.

1.2 State Attractors as Elements of a Markov Chain

A finite state Markov Chain of n attractors may be defined by a stochastic transition matrix P where

$$P = \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & & \bullet \\ \bullet & & & \\ \bullet & & & \\ \bullet & & & \\ P_{n1} & \bullet & \dots & P_{nn} \end{bmatrix} \quad (5)$$

and P_{ij} represents the probability of a transition from attractor i to attractor j . We define an attractor probability $V_i(t)$ as the probability that the system occupies state i after t discrete time units have elapsed. These time units are assumed to be large compared with the longest limit cycle in the state space. Based on the definition of $V_i(t)$, causality requires that

$$\sum_{i=1}^n V_i(t) = 1 \quad (6)$$

and

$$V_j(t+1) = \sum_{i=1}^n V_i(t) P_{ij} \quad \text{for } t = 0, 1, 2, \dots \quad (7)$$

We can combine the $V_i(t)$ components into an attractor probability vector $V(t)$. This gives the following compact representation for the noise-driven interaction between attractors.

$$V(t+1) = V(t)P \quad (8)$$

$$V(t) = V(0)P^t, \quad (9)$$

where P^t is defined as the t th power of the matrix P .

Therefore, the probability that the system occupies a given attractor after t time steps is given by postmultiplying the initial (or current) attractor probability distribution $V(0)$ by a power of the basic transition matrix P . Once the P_{ij} have been determined, only the initial probability distribution for attractors need be determined to completely specify the long-term response of the CA to a given level of added noise.

Progressively higher powers of the transition matrix will approach a limiting matrix P^∞ , if the system is characterized by time-independent probabilities.

The volume of state space traced out by a deterministic CA cannot grow with time. The rules that describe time-irreversible, or dissipative cellular automata (CA) contract, on average, the volume of state space traversed by the system dynamics. Therefore, the trajectory of a dissipative CA through state space must wither down to either a closed path or a single point. The limiting behavior is called an Attractor (A) or, alternatively, a Limit Cycle (L). The initial transient dynamics of a CA may be complex, but all initial points in state space must approach and finally reside on an attractor. The length of an attractor (L_i) is given by the number of states that make up the cycle. Deterministic CA generate attractors that are invariant under the dynamics and can neither cross nor be decomposed into other attractors. The transient paths that lead to an attractor are called Basins.

The strength of an attractor is measured by the volume of state space absorbed by the basin paths. It follows that a CA characterized by large

basins, i.e., a few strong attractors, requires relatively less information to be supplied to an initial cell pattern for the prediction of the final attractor observed. For example, a CA with a single attractor, such as Rule 255, given in Appendix A, requires no preselection of cell state patterns to predict with certainty the future dynamics of the automaton. However, rules including Nos. 18 and 22 are characterized by numerous attractors of various basin sizes. Predicting a particular limit cycle requires careful specification of the starting cell configuration in these latter examples.

From Information Theory, we can quantify the degree to which an attractor can be predicted from an initial starting point in state space as

$$\Omega_{SR} = \sum_{i=1}^n V_i \ln V_i, \quad (10)$$

where V_i is the volume of state space absorbed by attractor i , and n is the number of attractors in the space of a rule R , size S automaton. A more uniform measure of predictability, which normalizes Equation 10 to its maximum value for each rule type and automaton size, is given as

$$I_{SR} = \frac{\sum_{i=1}^n V_i \ln V_i}{\ln(n)}. \quad (11)$$

For a CA with more than one attractor, the value of I_{SR} ranges from zero (no information required to specify the final attractor) to unity (maximum information required to predict the final attractor). The future of a single-attractor CA is pattern-independent. I_{SR} for a single attractor is defined as zero.

2. Experiments

To measure the effect of noise on the dynamics of one-dimensional CA, we performed Monte Carlo tests on a large class of nearest-neighbor coupled automata. Due to the vast simulation space available, we restricted our

investigations to rule-homogeneous CA. The high-speed pipelined processors described in Section IV were used to conduct all the reported experiments.

The path of the dynamics through state space for a single simulation run is shown schematically in Figure 14. The experiment consisted of choosing an initial pattern of cell states, and then following the array dynamics until a repeating pattern, the attractor, was detected. After this initial attractor was recorded, a single, randomly selected cell state was flipped from 0 to 1 or vice versa. The new cell pattern was allowed to evolve to a second attractor. For arrays less than 16 cells in length the dynamics were studied for every possible starting configuration of cell states. Arrays longer than 15 cells were initialized by random state patterns to provide an unbiased sampling of the exponentially larger state space. For the larger array sizes we conducted 30,000 random samples of state space to accumulate reasonable statistics.

We analyzed the set of nearest-neighbor coupled CA rules to determine the minimum set of interactions required to span all behaviors. The complete set of canonical forms for these rules is given in Appendix A. Graphic examples of the transient dynamics of all 256 nearest-neighbor rules are given in Appendix B. Using the symmetry of periodic boundary conditions, we found 88 conjugate and reflection symmetric rule-groups. For example, reference to figures in Appendix B shows that Rules 14, 84, 143, and 213 lead to identical dynamics after reflection and/or conjugation of the observed patterns. As a result, a set of 88 rules completely defines the possible dynamics of one-dimensional, nearest-neighbor CA with periodic boundary conditions. We examined the dynamics of all 88 independent rules and array sizes between 3 and 28 cells in length.

3. Markov Results

Due to the large amount of reduced data compiled during our Markov simulations, we have divided the results and included them in this report as Appendices A through F.

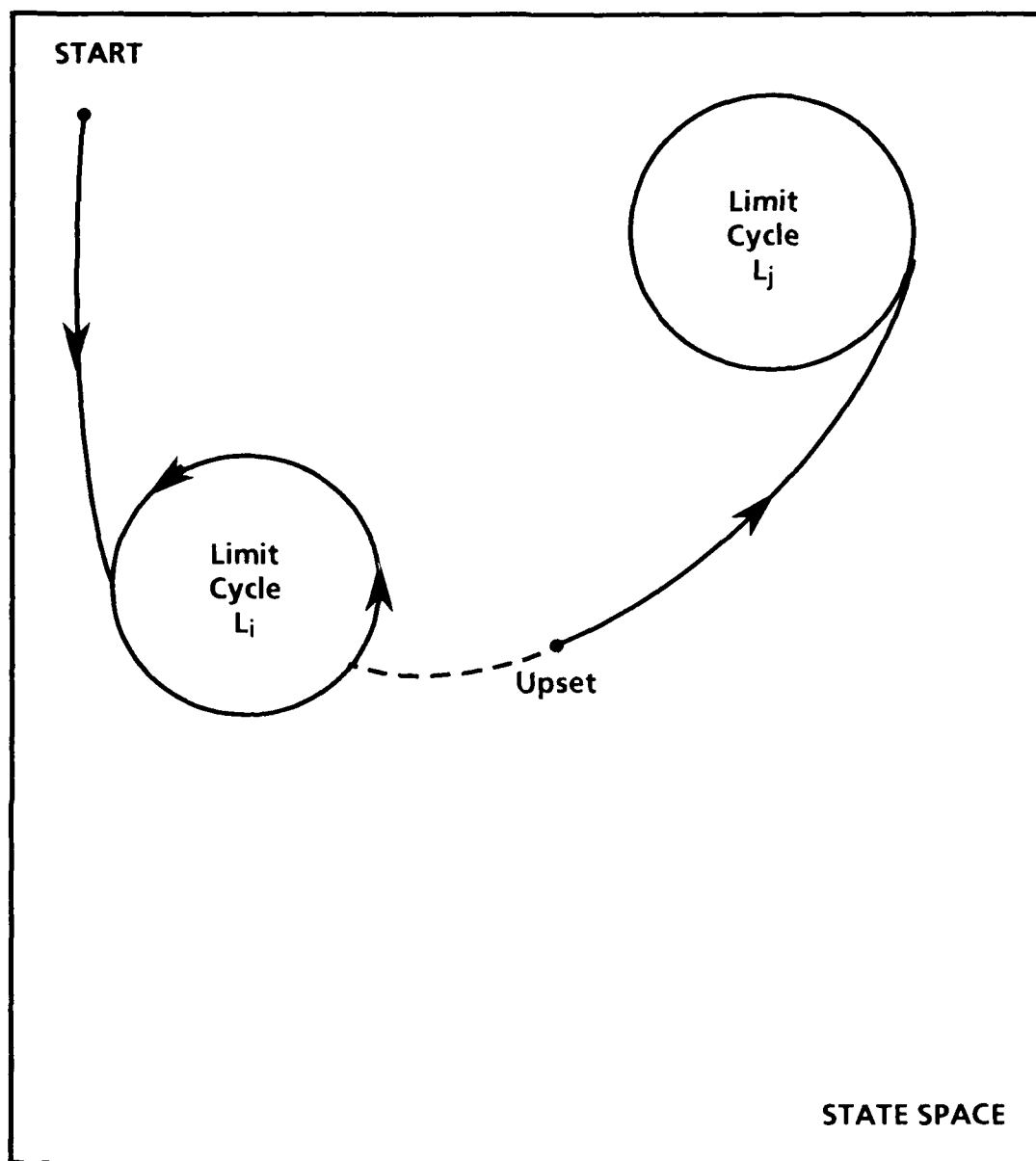


Figure 14. The effects of noise on the dynamics of nearest-neighbor coupled, one-dimensional CA.

VI. MULTIVALUED LOGIC IN QUANTUM COUPLED CIRCUITS

Sophisticated device behavior, grounded in the basic physics of nanometer-sized structures, could help reduce the number and range of device interconnects. Enhancing device sophistication is equivalent to embedding more states, and possibly more state interactions, into each active element. A scalable poly-state device could increase the density of stored data and control information. In principle, fewer elements, with fewer interconnections would be required to express the same function. This section describes our first steps in developing a multivalued logic system compatible with the basic characteristics of quantum coupled devices. Conventional multivalued algebra is reviewed, and new logic operators compatible with quantum devices are developed. Suggestions are made for possible embodiments of ternary logic gates in nanoelectronics structures.

1. Introduction

An inherent problem with ultra-integration will be the further decrease in our ability to access directly a given functional resource at all levels of complexity. For a minimum feature size ($1/S$), resources may grow as fast as S^2 and S^3 for two- and three-dimensional circuits, respectively. However, on-chip access methods are essentially peripheral, so the dimensionality of I/O access may always be at least one dimension behind component topology. One approach to reducing I/O bottlenecks, while avoiding interconnect-intensive architectures, would be to develop sophisticated devices that perform more complex logic functions than transistor-like switching. Increasing device complexity amounts to extracting greater functionality from known physical phenomena. For example, Josephson Junction (JJ) technology allows an A/D convertor to be constructed using one JJ device per digitized bit.⁸ This significant improvement in functional density arises not from the cleverness of the device architecture, but simply from the physics of very small, very cold things. Ideally, increased complexity should arise naturally from the basic physical properties of the device technology. In our research we considered a method for increasing device complexity by developing a multivalued logic system based on quantum device properties.

1.1 Advantages of Multivalued Logic Approaches

There are technologically independent arguments suggesting that 2.718 (Napierian basis) represents the optimal switching network radix.¹⁶ Thus, on theoretical grounds, 3-valued, or radix-3 arithmetic should be preferred over binary. However, most abstract analyses neglect practical technological constraints. The cost of multivalued (MV) logic elements will be very dependent on the gate technology. To date, there have been insufficient practical advantages to justify the use of high-radix logic systems over the simplest, 2-valued standard. Realistic performance comparisons between actual M-valued hardware and its binary counterparts must assume a common technology base.

The most important benefit offered by MV logic is the potential to reduce interconnect complexity by embedding more states, and more functionality, within the same number of switching elements. Analysis shows that even a cost-inferior M-valued system may win out over a binary scheme when scaling problems undermine performance of 2-valued networks. For example, ternary (3-valued) multipliers have been conceived that contain 60% fewer interconnects and 20% fewer devices than do equivalent binary circuits.¹⁷ MV logic systems trade device count for device complexity. With scaledown, the savings in interconnect area can outweigh the cost of increasing device complexity.

2. Theory of Multivalued Algebra

To investigate the potential for utilizing MV logic elements in ultrascaled circuits, we first review the basics of MV algebra. Consider the general multivalued, N-input, single-output logic gate shown in Figure 15.

The gate realizes physically some function F of the input values I_i . For simplicity, we assume that the possible input and output values belonged to the same alphabet Q such that,

$\{Q\} = \{E_1, E_2, \dots, E_M\}$	Symbol alphabet and I/O definitions
Inputs $I_1..I_N \in \{Q\}$	for an M-valued, N-input
Output $S \in \{Q\}$	logic function F .
$S = F(I_1, I_2, \dots, I_N)$	

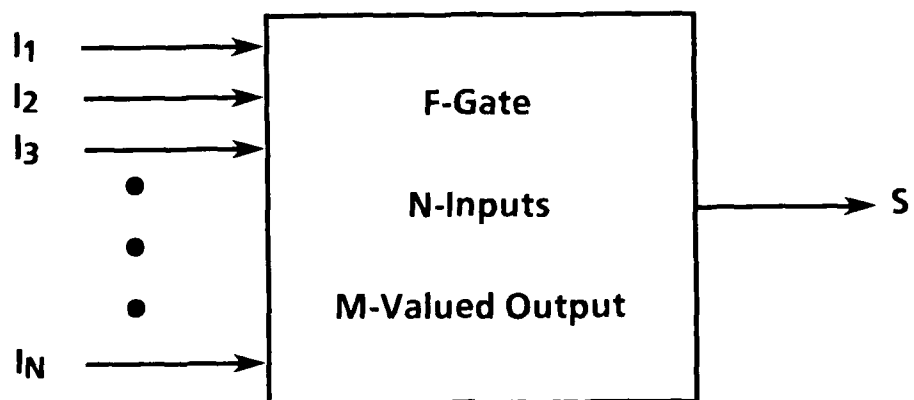


Figure 15. General M-valued, N-input logic gate.

where the E_i are the valid symbols that can appear on any line attached to our hypothetical gate. For example, a 2-valued propositional calculus might define E_1 as "T" for "True" and $E_2 = "F"$ for "False." In binary switching theory the two symbols E_1 and E_2 are usually labeled "0" and "1," respectively. In a physical structure these symbols would correspond to measurable values such as steady state voltage or current amplitudes.

The number of different multivalued functions of N -input variables was easily determined. We defined a function space $[M,N]$ as the set of all possible deterministic functions that could be realized by the multivalued, N -input gate in Figure 15. In general, there are M^N different combinations of symbols that can be applied to the gate input lines. When specifying the internal, deterministic function F , we may associate any one of the M alphabetic symbols E_i with each possible input combination, $\{I_1, I_2, \dots, I_N\}$. The total number R of unique functions that *could* be defined for a multivalued, N -input, single-output gate is equal to:

$$R = (M)^L$$

Size of the function space $[M,N]$

where $L = M^N$ for an M -valued, N -input, single-output logic gate.

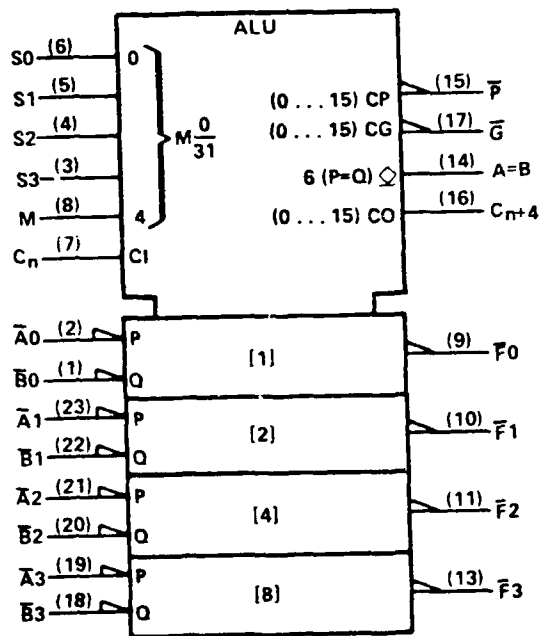
Table 2 shows how the size of $[M,N]$ -space varies with M and N .

2.1 Functional Decomposition in Multivalued Logic

Table 2 shows how increasing the number of allowed symbols within a logic system can dramatically increase the function space for the algebra. One consequence of the vast functional territory of M -valued, N -input logic, is that it is impractical to construct unique device types for all but a small fraction of multivalued logic functions. For example, the modern arithmetic logic unit (ALU) represents a sophisticated gate that can be programmed to perform many of the more useful functions of two or more inputs. A popular 2-valued ALU is shown in Figure 16. Although quite useful, this building block provides directly only 32 of the 16^{512} possible functions within the $[2,9]$ -function space of the system. Even the collection of gates shown in Figure 17 provides directly only 12 of the 16 possible single-output binary functions of $[2,2]$ -space.

Table 2. Function Space for Various Numbers of Inputs (N) and Output States (M)

N \ M				
	2	3	4	5
1	2	27	256	3,125
2	16	19,683	4×10^9	3×10^{17}
3	256	7×10^{12}	3×10^{38}	2×10^{87}
4	65,536	4×10^{38}	4256	51024



SELECTION					ACTIVE HIGH DATA	
					M = H LOGIC FUNCTIONS	M = L, ARITHMETIC OPERATIONS
S3	S2	S1	S0			
L	L	L	L	F = A	F = A	F = A PLUS 1
L	L	L	H	F = A + B	F = A + B	F = (A + B) PLUS 1
L	L	H	L	F = A + B	F = A + B	F = (A + B) PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	F = A + B	F = A PLUS A B	F = A PLUS A B PLUS 1
L	H	L	H	F = B	F = (A + B) PLUS A B	F = (A + B) PLUS A B PLUS 1
L	H	H	L	F = A - B	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	F = A B	F = A B MINUS 1	F = A B
H	L	L	L	F = A + B	F = A PLUS A B	F = A PLUS A B PLUS 1
H	L	L	H	F = A - B	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = R	F = (A + B) PLUS A B	F = (A + B) PLUS A B PLUS 1
H	L	H	H	F = AB	F = A MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A*	F = A PLUS A PLUS 1
H	H	L	H	F = A + B	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

Figure 16. Logic symbol and function table for a popular (LS181) ALU module.



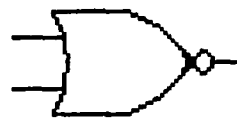
AND



NAND



OR



NOR



XOR



XNOR

Figure 17. Six popular [2,2]-space logic functions.

Since direct realization of arbitrary multivalued functions is, at best, technically prohibitive, attempts are made to discover minimum sets of functions that can be combined and/or sequenced to represent any required function. Fortunately, P-valued logic functions of two input variables exist that can be combined to express any M-valued function of N variables, for any value of P greater than or equal to M.

2.2 Well Known M-Valued Algebra

Several popular logic systems have been discovered that are capable of expressing any M-valued function of N variables. Necessary and sufficient conditions for functional completeness are also known.¹⁸ Perhaps the most studied of the complete sets of M-valued algebras is defined by the following operators:

Popular M-Valued Logic Operators

Operator	Operation	Function F	Equation
Min	$I_1 \text{ Min } I_2$	= Minimum of $\{I_1, I_2\}$	11a
Max	$I_1 \text{ Max } I_2$	= Maximum of $\{I_1, I_2\}$	11b
Cycle	$I_1 I_2$	(I_1 plus I_2) module M	11c
Complement	$I_1 \sim$	(M-1) minus I_1	11d
Match	$\langle I_1, E_j \rangle$	E_M if $I_1 = E_j$	11e
Match	$\langle I_1, E_j \rangle$	E_1 otherwise	11e

where

M = number of symbols in alphabet,

I_1 and I_2 are inputs,

F is the function result, and

$E_j = j-1$, so that $E_i < E_j$ for any $i < j$.

Monotonic definition of (12)

M-valued alphabet Q.

For simplicity, we defined and ordered the symbol alphabet Q so that direct arithmetic operations can be used to comprehend the function of each operator. Tables 3 and 4 list the "truth" tables of these operators for binary- and ternary-valued algebra, respectively. Inspection shows that the

Table 3. [2,2]-Space Operator Set for Conventional Boolean Algebra

Inputs		"Min"	"Max"	"Cycle"	"Complement"		"Match"	
I ₁	I ₂	I ₁ min I ₂	I ₁ max I ₂	I ₁ I ₂	I ₁	I ₁ ~	I ₁	X <I ₁ :X>
0	0	0	0	0	0	1	0	1
0	1	0	1	1	1	0	0	1
1	0	0	1	1			1	0
1	1	1	1	0			1	1

Table 4. [3,2]-Space Operator Set for Conventional "Min" - "Max" Algebra

Inputs		"Min"	"Max"	"Cycle"	"Complement"		"Match"	
I ₁	I ₂	I ₁ min I ₂	I ₁ max I ₂	I ₁ I ₂	I ₁	I ₁ ~	I ₁	X <I ₁ :X>
0	0	0	0	0	0	2	0	2
0	1	0	1	1	1	2	0	1
0	2	0	2	2	2	2	0	2
1	0	0	1	1			1	0
1	1	1	1	2			1	1
1	2	1	2	0			1	2
2	0	0	2	2			2	0
2	1	1	2	0			2	1
2	2	2	2	1			2	2

2-valued operators of Min, Max, Cycle, Match, and Complement are similar, respectively, to the familiar binary logic functions of AND, OR, EXOR, EXNOR, and NOT. The "Max" and "Min" operators produce, respectively, the greater or lesser of the two input variables I_1 and I_2 . It is interesting that the 2-valued "AND" operator of *only if both* is more generally represented by the *lesser of* function "Min". The unary Match operators perform "symbol test" on single variables and could be recast using the Dirac delta function as,

$$\begin{aligned} \text{Match } \langle I_1; E_j \rangle &= (M-1) \text{ times } \delta_{1j} && \text{Match Operator} && (13) \\ \text{where } \delta_{1j} &= 1 && \text{if } I_1 = E_j && \text{in terms of Dirac} \\ &= 0 && \text{if } I_1 \neq E_j && \text{Delta functions} \end{aligned}$$

The Identity symbols for this set are $E_1=\{0\}$ and $E_M=\{M-1\}$. It is easily shown that the operator set of Equation (11) satisfies the following relationships when applied to the two input variables X and Y ,

Idempotence:	$X \text{ Min } X = X$ $X \text{ Max } X = X$
Absorption:	$X \text{ Max } (X \text{ Min } Y) = X$ $X \text{ Min } (X \text{ Max } Y) = X$
Null Symbols:	$X \text{ Min } E_M = X$ $X \text{ Max } E_1 = X$
Universal Operation:	$X \text{ Min } E_1 = E_1$ $X \text{ Max } E_M = E_M$
DeMorgan's Laws:	$X \text{ Min } Y = (X \sim \text{Max } Y \sim) \sim$ $X \text{ Max } Y = (X \sim \text{Min } Y \sim) \sim$
Commutativity:	$X \text{ Max } Y = Y \text{ Max } X$ $X \text{ Min } Y = Y \text{ Min } X$

The combination of Equations (11a) and (11b) satisfies the distributivity condition:

Distributivity:

$$X \text{ Max } (Y \text{ Min } Z) = (X \text{ Max } Y) \text{ Min } (X \text{ Max } Z)$$

$$X \text{ Min } (Y \text{ Max } Z) = (X \text{ Min } Y) \text{ Max } (X \text{ Min } Z)$$

It is also straightforward to prove that the set of operators defined by Equation (11) is functionally complete. In particular, a canonical form for any M-valued function can be written as

$$F = \sum_{k=0}^{R-1} \left[S_k \text{ Min } \prod_{j=1}^N \langle I_j : C_{kj} \rangle \right] \quad \begin{array}{l} \text{Sum-of-products (14)} \\ \text{form of any} \\ \text{M-valued function F.} \end{array}$$

$$\text{where } \prod_{j=1}^N a_j = a_1 \text{ Min } a_2 \text{ Min } \dots \text{ Min } a_N$$

$$\text{and } \sum_{k=0}^{R-1} b_k = b_0 \text{ Max } b_1 \text{ Max } \dots \text{ Max } b_{R-1}$$

The S_k are the function outputs associated with each of the R input combinations $\{I_1..I_N\}$. The C_{kj} are defined by

$$k = C_{k1} + MC_{k2} + M^2C_{k3} + \dots M^{N-1}C_{kN} \quad \begin{array}{l} \text{Defining equations} \\ \text{for the } C_{kj}. \end{array}$$

Equation (14) demonstrates that the proper choice of constants, Match, Min, and Max functions allows any M-valued function to be decomposed into [M,2]-space logic operations. As a concrete example, we show in Table 5 a randomly selected [3,2]-space function. Also shown is the combinatorial form for this function in terms of the operator set of Equation (11). Due to the arithmetic definition of the M-valued alphabet, the combinatorial form in Table 5 does not need to take explicit account of those terms that evaluate to $E_1=0$.

2.3 Completeness and Practicality in M-Valued Circuits

The functional completeness of each of the binary logic systems {NAND}, {NOR}, {Majority-Vote}, and {AND, EXOR} is well known. Any one of these sets of mathematical operations is sufficient to express any arithmetic operation,

Table 5. Truth Table and Canonical Form for Randomly Selected [3,2]-Space Function

k	I ₁	I ₂	S _k =F(k)
0	0	0	0
1	0	1	0
2	0	2	1
3	1	0	1
4	1	1	0
5	1	2	0
6	2	0	2
7	2	1	0
8	2	2	2

$$\begin{aligned}
 F = & [0 \min (I_1:0> \min <I_2:0>)] \max \dots, \\
 & [0 \min (I_1:0> \min <I_2:1>)] \max \dots, \\
 & [1 \min (I_1:0> \min <I_2:2>)] \max \dots, \\
 & [1 \min (I_1:1> \min <I_2:0>)] \max \dots, \\
 & [0 \min (I_1:1> \min <I_2:1>)] \max \dots, \\
 & [0 \min (I_1:1> \min <I_2:2>)] \max \dots, \\
 & [2 \min (I_1:2> \min <I_2:0>)] \max \dots, \\
 & [0 \min (I_1:2> \min <I_2:1>)] \max \dots, \\
 & [2 \min (I_1:2> \min <I_2:2>)]
 \end{aligned}$$

Canonical Form

including, of course, the operations required of general-purpose computers. Within each system, each logic operation represents a necessary component of a complete algebra. However, a computer based solely on one of these sets would be only a laboratory curiosity. Practical computers utilize an over-specified set of functionally complete logic operations to maximize the functional density of a desired circuit. For example, it is neither necessary nor cost-effective to use a two-input EXOR gate to serve the function of a simple inverter (Unary Complement). Therefore, a repertoire of more-than-sufficient operators, {AND, NAND, NOR, OR, NOT, EXOR, EXNOR}, is used in the design of actual logic circuits. Similar arguments can be made for the need to employ more-than-sufficient MV logic components.

2.4 Overspecification in M-Valued Algebra

Neither the "Cycle" nor the "Complement" operator appears in the canonical form of Equation (14). Therefore, the operator set of Equation (11) is overspecified in the sense that not all of the operators are essential to the representation of combinatorial functions.

Moreover, our examination of Equation (14) showed that even the operators that were used to compose the canonical form are themselves overdefined. To study operator overspecification, we employed four hypothetical operators labeled "&", "#", "@", and "%" and rewrote Equation (14) as

$$F = \sum_{k=0}^{R-1} \left[S_k @ \prod_{j=1}^N (I_j \% C_{kj}) \right] \quad \text{Sum-of-products (15)}$$

form of any M-valued function F, using operators @, %, &, and #.

$$\text{where } \prod_{j=1}^N a_j = a_1 \& a_2 \& \dots \& a_N$$

$$\text{and } \sum_{k=0}^{R-1} b_k = b_0 \# b_1 \# \dots \# b_{R-1}$$

We found that since the Match operator can only produce a value of E_1 or E_M , the new "&" operator need not be defined for any input combination that contains any symbols other than E_1 and E_M . By analogy with Equation (14), we defined the "&" operator as simply the original Min function, but with arbitrary "don't cares" associated with those input terms that could not normally occur. By using this definition for "&," the "#" function could be similarly defined as the Max operator with suitable "don't care" substitutions for its irrelevant input cases. Similar reasoning was used to specify the @ operator. Lastly, we redefined the Match operator as a more general two-input operator "%," so that two-input variables can be tested for equality. The final set of the new operators for [3,2]-space is shown in Table 6. It is noteworthy that the "@" operator can replace the "&" function if we choose particular values for some of the arbitrary outputs of the "&" operator. Using the occurrence of "don't cares" in the internal workings of the canonical form, several other sets of sparse, functionally complete operator sets may be devised.¹⁸ These results demonstrate how little structure is required to form combinatorially complete algebras. We used this new algebra to prove the functional completeness of simple multivalued quantum coupled circuits.

3. M-Valued Logic in Quantum Coupled Circuits

3.1 Navigating in a Quantum Mechanical World

As interwell dimensions scale to the order of the Bloch wavelength (roughly 300 Å in many semiconductors), the confinement of electronic charge becomes difficult. As discussed in Section III, the scaledown of physical dimensions causes an unavoidable coupling between devices that store or manipulate the flow of charge. The leakage of electronic charge from a potential well is called *electronic tunneling*.¹¹ For this reason devices, such as transistors, which rely on potential barriers to gate, direct, or confine electric charge, cannot scale below roughly 0.1 μm .¹⁹

It is apparent that the nanometer-sized world is rather nonclassical, since electronic boundaries tend to become porous and electrons become claustrophobic. However, classical laws are not entirely ignored in ultra-scaled structures. In particular, the conservation laws of energy and momentum still govern the changes in the average electron position and

Table 6. Combinatorially Complete Set of Ternary Logic Operators. Entries marked "X" may be arbitrarily chosen as 0, 1, or 2.

Inputs		"%"	"&"	"@"	"#"
I ₁	I ₂	I ₁ % I ₂	I ₁ & I ₂	I ₁ @ I ₂	I ₁ # I ₂
0	0	2	0	0	0
0	1	0	X	0	1
0	2	0	0	0	2
1	0	0	X	X	1
1	1	2	X	X	X
1	2	0	X	X	X
2	0	0	0	0	2
2	1	0	X	1	X
2	2	2	2	2	X

energy. For example, an electron can easily tunnel into a neighboring "quantum coupled" well only if its initial and final well energies are the same. Energy is easy to conserve in tunneling between identical quantum wells because symmetry requires the same allowed energies in either well. However, as shown in Figure 5 (Section II), the tunneling probability between dissimilar wells is determined by the relative position of the energy levels within the two wells.

3.2 Multivalued Quantum Logic

One of the more intriguing features of spatial quantization of charge provides us the ability to engineer several sharply defined energy states within the same quantum well. To first order, the number of distinct energy levels within a well is inversely proportional to the square of the well size. The relative energy of the well states can be controlled by the careful selection of semiconductor materials, or by using mixed-material, superlattice techniques. Thus, materials science and lithography can be used to adjust the number and energy of the resonant energy levels distributed among a collection of quantum structures. The energy resonance or dissonance between well states controls the strength and direction of charge flow.

Quantum wells are, in principle, very simple physical structures. Since quantum wells need not be formed near a ground plane, it is possible for the charge in a neighboring well to define in part the potential energy of a "floating well." This electrostatic coupling can be thought of as the input terms of a logic function carried out by the modulated resonant tunneling structure. The electrostatic potential of a quantum well will also depend on the net charge bound by the well. However, we will neglect such self-consistency effects here.

3.3 Ternary Quantum Logic Structures

Quantum devices can be constructed such that each physical structure contains one or more discrete energy states. The energy diagram for an abstract, three-dimensionally quantized quantum well or dot might be as that shown in Figure 18. If we neglect self-consistency effects, the energy of charge placed into the well remains relatively constant until the lowest energy level is filled. Above this capacity, new charge will reside in the

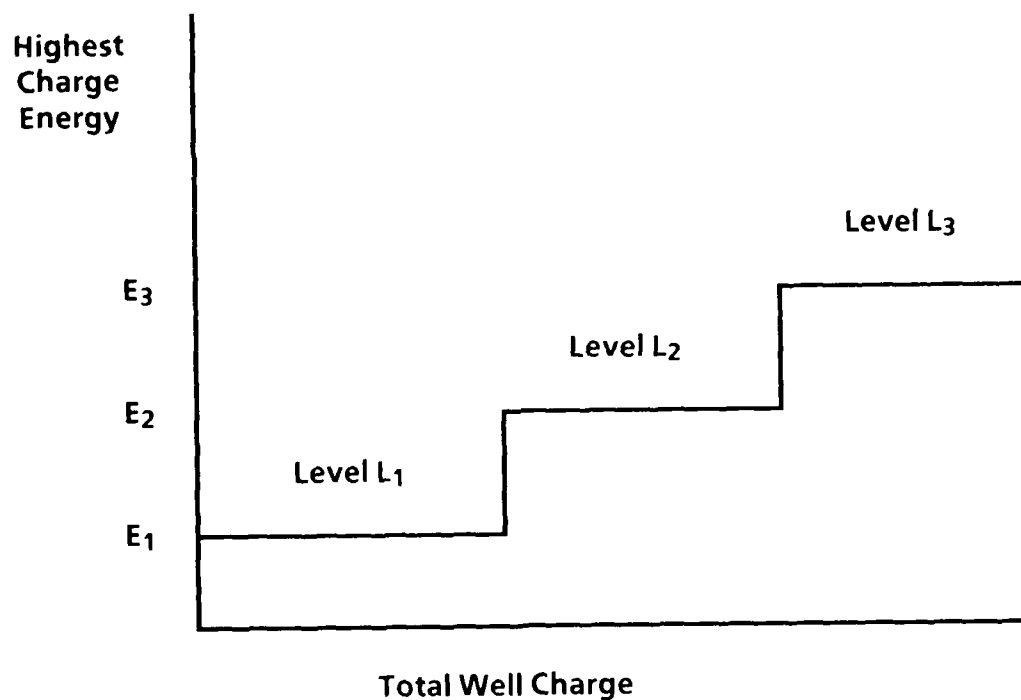
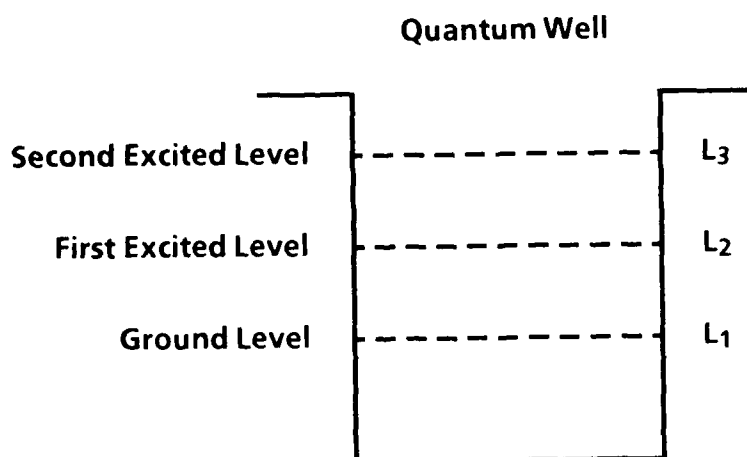


Figure 18. Quantum jumps in charge energy due to quantum size effect.

next highest energy state, as taught by the Pauli Exclusion Principle. Now assume that we have defined our device structure such that the addition of two units of charge just fills the lowest energy level. If we equate the level of charge contained in a quantum well with a measure of information stored in the structure, then the physical value of our multivalued alphabet as defined by Equation (12) can be expressed physically as:

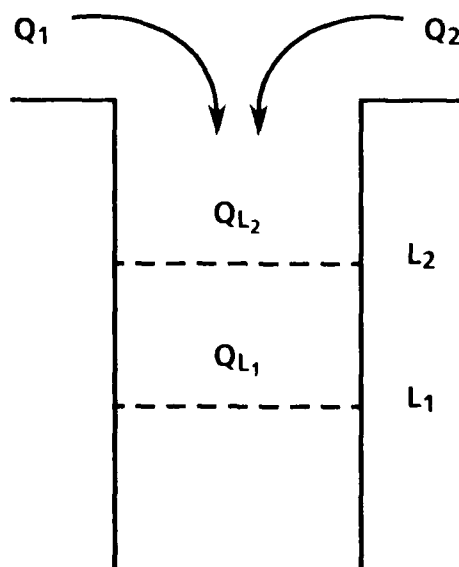
$E_1 = 0$ units of well charge	Representation of an
$E_2 = 1$ unit of well charge	M-valued alphabet by
$E_3 = 2$ units of well charge	quantum well charge.

The logical state of the well can be expressed as a function of the charge carried by the energy levels of the well.

Consider the case where the well charge is obtained by absorbing the charge from two adjacent "input" wells. For various quantum well widths, several distributions of charge over the well energy levels may result. The charge distribution for several well types is shown in Table 7. It is seen from the table that the quantity of charge contained in the second level of well type B represents the *minimum* of the two added charge packets, Q_1 and Q_2 , except for the case where $Q_1 = Q_2 = 1$ unit of charge. Similarly, the charge accumulated in the first level of well type B is equal to the maximum value of the two charge inputs, Q_1 and Q_2 , except for the case where $Q_1 = Q_2 = E_1 = 1$. In this isolated case the charge densities in Levels L_1 and L_2 represent the arithmetic sum and difference of Q_1 and Q_2 , respectively. Thus, the most natural distribution of charge within a quantum well nearly emulates the traditional "Min" and "Max" MV operators. It would seem that some additional effort is required if we are to create quantum well interactions that emulate the familiar M-valued calculus. Fortunately, the "&," "#," and "@" operators defined by Equation (15) precisely predict the value of the charge held by both the ground state and excited state energy levels of well type B for all combinations of Q_1 and Q_2 . Therefore, the addition of two quantized, M-valued packets of charge to a simple quantum well can naturally express the "&," "#," and "@" operations on [3,2]-space variables.

Table 7. Result of Adding Charge Units Q_1 and Q_2 to Four Types of Quantum Wells. [Entries Q_{L1} and Q_{L2} are the final values of charge located in quantum levels L_1 and L_2 , respectively.]

INPUTS		WELL A		WELL B		WELL C		WELL D	
Q_1	Q_2	Q_{L1}	Q_{L2}	Q_{L1}	Q_{L2}	Q_{L1}	Q_{L2}	Q_{L1}	Q_{L2}
0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	1	0	1	0
0	2	1	1	2	0	2	0	2	0
1	0	1	0	1	0	1	0	1	0
1	1	1	1	2	0	2	0	2	0
1	2	1	2	2	1	3	0	3	0
2	0	1	1	2	0	2	0	2	0
2	1	1	2	2	1	3	0	3	0
2	2	1	3	2	2	3	1	4	0



The quantum phenomenon of resonant tunneling can also be utilized to implement the modified "Match" operator, %. To demonstrate this, we place two electrically floating resonant tunneling wells (W_1 and W_2) in close proximity to two input wells (W_3 and W_4) as shown in Figures 19 and 20. The electrostatic potentials of the input wells are used to define the potential energy of the floating tunnel wells. These floating wells will be in tunnel resonance if, and only if, the charges carried by the input wells are identical. As shown in Figure 20, any dissimilarity between input well charges will frustrate the resonance condition between the tunneling wells. To complete the function layout, the tunnel wells are linked to a large source of charge, W_5 , which can flow across the wells when in resonance. This charge feeds into a final quantum well that has been lithographed to hold a maximum of E_M units of charge. This final well represents the output of the "%" operation carried out by the structure described. After a "calculation", the "%" well, W_6 , would remain empty if $Q_1 = I_1$ was not equal to $Q_2 = I_2$, and absorb $E_M = 2$ units of charge when $I_1 = I_2$.

4. Additional Note

It can be argued that some form of nonlinear operation is a necessary condition for computational completeness. At the least, the readout of the state of a computing device requires some form of thresholding operation to discriminate between the possible valid output symbols. It is probably also true that nonlinear operations are required within a complex or chained computation to filter noise. However, it is not necessary for the computational events themselves to be logically or thermodynamically irreversible. For this reason, purely quantum mechanical computations are possible in principle. In the quantum coupled systems described above, the "nonlinear" operations arise naturally from the energy quantization of quantum well states. If a system were purely linear, then the principle of superposition or simple proportion should apply to the various symbolic representations of the internal machine state. However, the Pauli Exclusion Principle guarantees that, at some point, the description of the state of two quantized levels of charge (information) must diverge. For this reason, linearity does not apply fully to a quantum system which in fact utilizes more than one energy state in performing calculations.

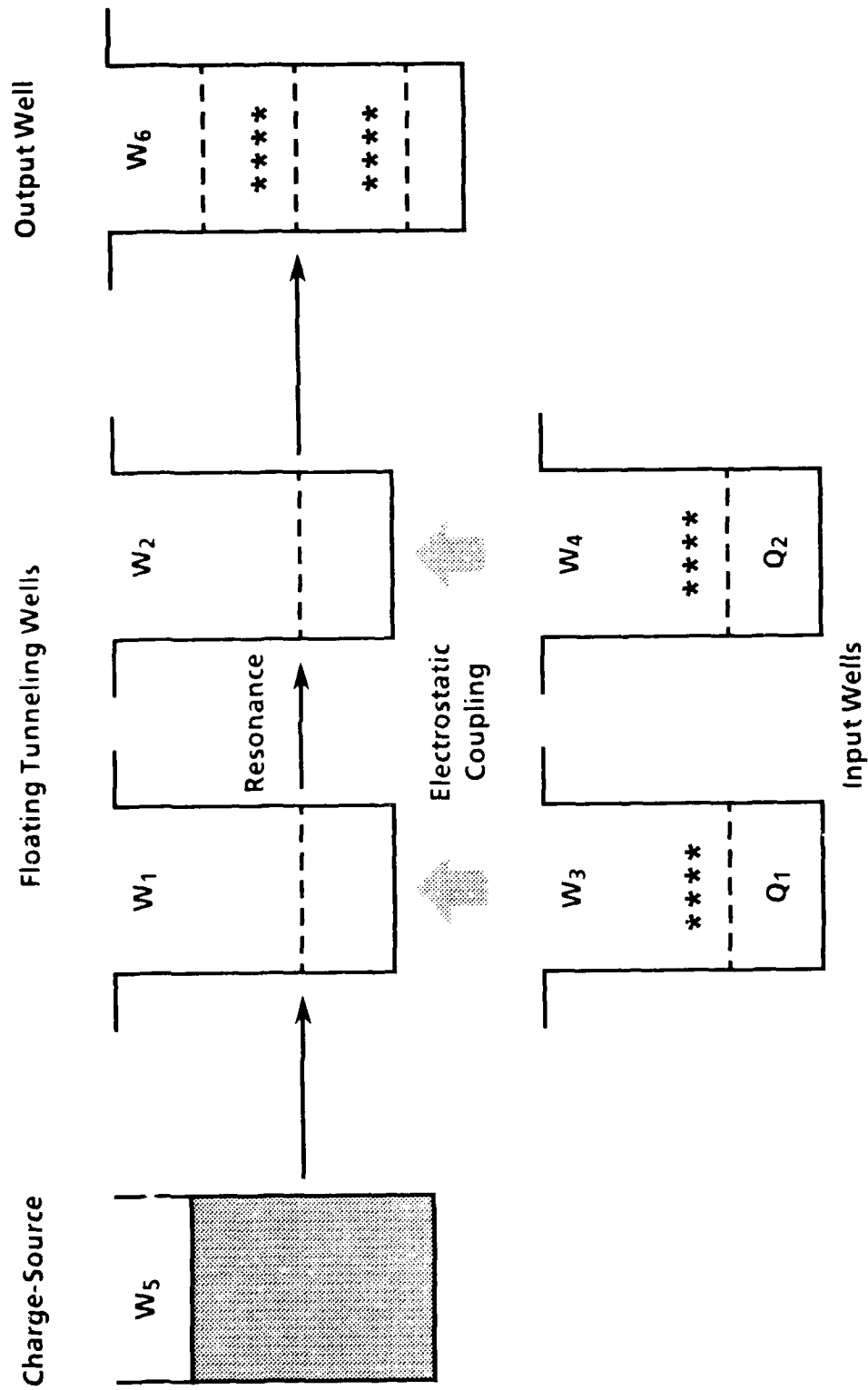


Figure 19. Construction of "%" operator, using quantum well structures. Charge in input wells W3 and W4 modulates energy of tunnel wells W1 and W2 to control charge transfer between source well W5 and output well W6. Resonant coupling condition is shown.

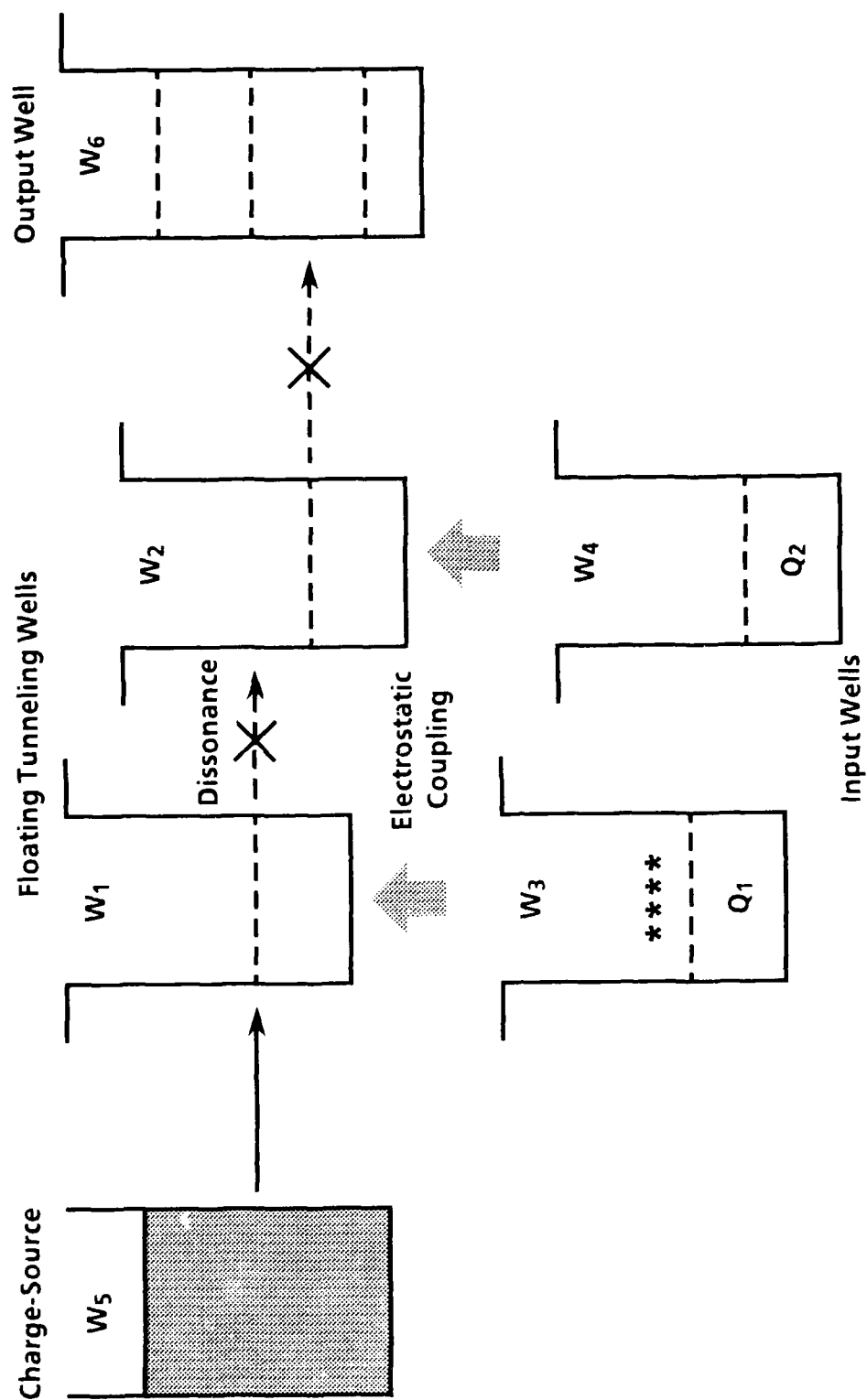


Figure 20. Construction of "%" operator, using quantum well structures. Charge in input wells W_3 and W_4 modulates energy of tunnel wells W_1 and W_2 to control charge transfer between source well W_5 and output well W_6 . Off-resonance condition is shown.

5. Summary

We have shown in principle that multivalued logic operations may be embodied by quantum-coupled devices in a straightforward manner. Traditional "Min"- "Max" multivalued algebra is not immediately compatible with the simplest interactions between quantum coupled device structures. For this reason, it is not certain whether the large amount of prior work on, e.g., the minimization of multivalued logic systems, can be applied to the efficient design of more complex quantum MV logic circuits.

VII. SELF-TIMED CELLULAR AUTOMATA

To maintain scalability, the use of on-chip wiring must be avoided in nanometer-sized computing functions. This requirement introduces a major dilemma in function design: it will no longer be possible to broadcast synchronizing clocks to individual logic elements. This section discusses an approach for organizing cellular automata computations in terms of locally clocked cell interactions.

1. Introduction

The physical scaledown of electronic circuits exacerbates many design problems associated with low-level connectivity and timing. Of particular concern is the problem of clock skew within and between functions. As circuit complexity increases, some form of mutual signaling must be used to synchronize independent functions. These clock signals provide an orderly method for data exchange between asynchronous functions and also supply the "arrow of time" needed to direct the flow of data within functions. It is essential to devise generic methods for distributing skew-free clocks, or to provide self-timing techniques that are compatible with nanometer-sized circuitry.

We have suggested that the cellular automaton (CA) architecture permits function scalability into the nanometer regime. A CA structure avoids interconnect saturation at low levels of functionality and encourages modular approaches to function decomposition. However, the low dimensionality of the CA places obvious limits on the degree to which the activities of large device and function arrays may be synchronized. We have suggested that optical wavefronts may be used to broadcast skew-free clock signals to electro-optically active quantum devices.²⁰ If optical or similar global clocking methods are found to be unsuitable, then it will be difficult to exploit the density advantages of nanoelectronic CA unless some measure of asynchronous computation is possible at the lowest levels of functionality.

In this section we discuss one method for managing the timing of computation in cellular automata. The approach is most easily applied to nearest-neighbor CA, although any level of connectivity can be supported at the expense of cell complexity and computational throughput. We emphasize

the use of the method for nearest-neighbor CA because this class of CA is known to be functionally complete.

2. Model for Asynchronous Cellular Arrays

In a cellular automaton the future logic state of a given cell is a deterministic function of its present state and that of its immediate neighbors. For our model we assumed that the operations carried out within each cell were synchronous, and that all intercell communications were asynchronous. In particular, the calculation of the next states of a cell was assumed to be a synchronous internal cell operation, while asynchronous cell operations were taken to include both the input of neighboring cell values and the broadcast of new states to local cell neighbors. These suppositions should be reasonable as long as cell complexity and physical cell size are both small.

Our plan for coordinating intercell signaling is based on a three-state cycle. Seitz has described two- and four-cycle asynchronous systems.²¹ We developed a three-cycle method based on the requirements discussed below.

First, it is assumed that the scaling advantages of the CA architecture will have the greatest impact on the lowest levels of functionality and feature size. As minimum geometries decrease into the nanometer regime, there will be a commensurate reduction in the available functional complexity for communication and control. Thus, minimizing the complexity of the synchronization process is essential. We equate this reduction of complexity to a minimization of the number of states that must be allocated for the synchronization of nanoevents.

Second, we can show that a two-cycle synchronization method is not adequate for classical CA. In a conventional digital circuit, the "arrow-of-time" is provided naturally by the flow of information processing from input lines, through logic gates, to output lines. In a CA the distinction between input lines and output lines is only one of semantics. The output of a cell at time (t) becomes one of its own inputs at time ($t+1$). Therefore, it is necessary to encode a cell state with enough information to distinguish an

"input" from an "output" event. We found that three sub-cycles or phases were sufficient to fulfill these requirements.

Cell activities (input, process, and output) were assigned to each phase of a periodic, three-phase clock as shown in Figure 21. Figure 22 depicts a high-level representation of the sophistication required to implement the self-timed system. For clarity, we discuss the timing process from the perspective of a single cell, called cell X. As shown in Figure 22, each cell exports to other cells only a partial representation of its logic state. Neighboring cells receive the timing status of cell X, as well as the fraction of the cell state shown schematically as register Z. Part of the state of a cell (register Y) is not directly accessible by other cells. Within the context of a classical CA, the inaccessibility of cell information is taken into account in the definition of the cell interaction rules. Cell operations within each clock phase break out as follows.

- Phase 0: Referring to Figures 21 and 22: each cell attempts to update an internal representation of its next state during phase 0. A next-state calculation is assumed to be valid only if all neighbors are in either phase 0 or phase 1. Therefore, cell operations within phase 0 are inhibited by any neighborhood cells which reside in a phase 2 condition. When all cell neighbors exit the phase 2 condition, the new state of cell X is calculated and stored in state register Y. The clock phase of cell X is then incremented to phase 1.
- Phase 1: Cell X has performed the internal processing of its inputs in phase 0. No processing operations are carried out by cell X while in the phase 1 sub-cycle. Neighboring cells are alerted to the completion of phase 0 processing activities when the phase 1 sub-cycle is detected. A cell in phase 1 may not update its external state, since all neighbors may not have completed their processing of the existing state patterns. The transition of a cell out of phase 1 into phase 2 occurs only when no neighboring cell resides in a phase 0 sub-cycle.

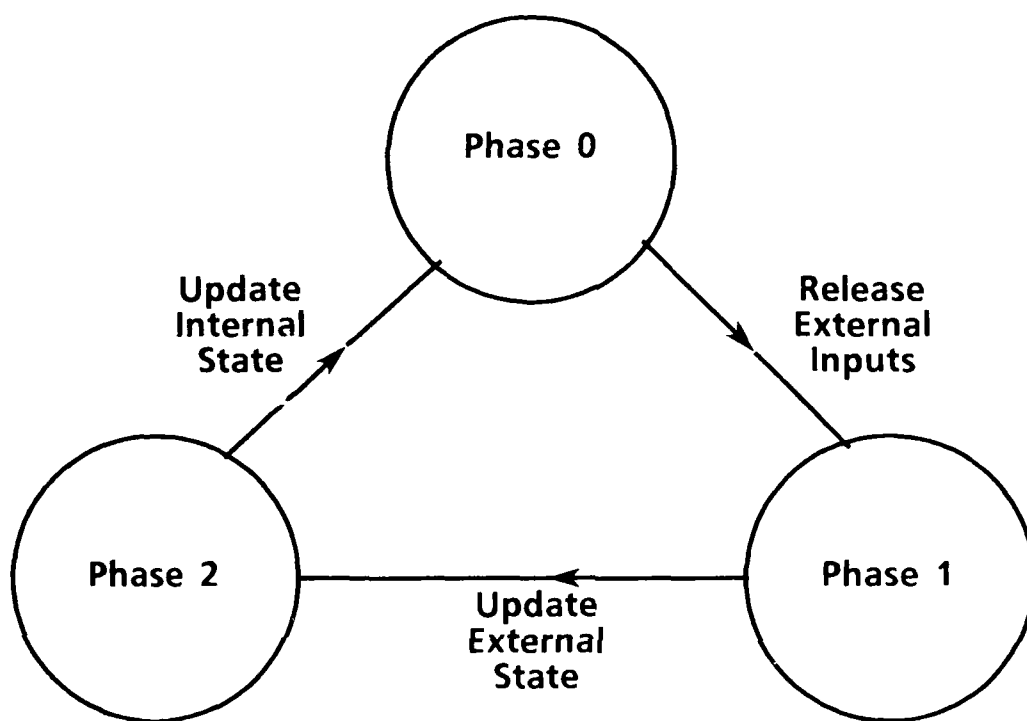


Figure 21. Transition graph for asynchronous automata clock phases and cell operations.

A Single Asynchronous Automaton Cell X

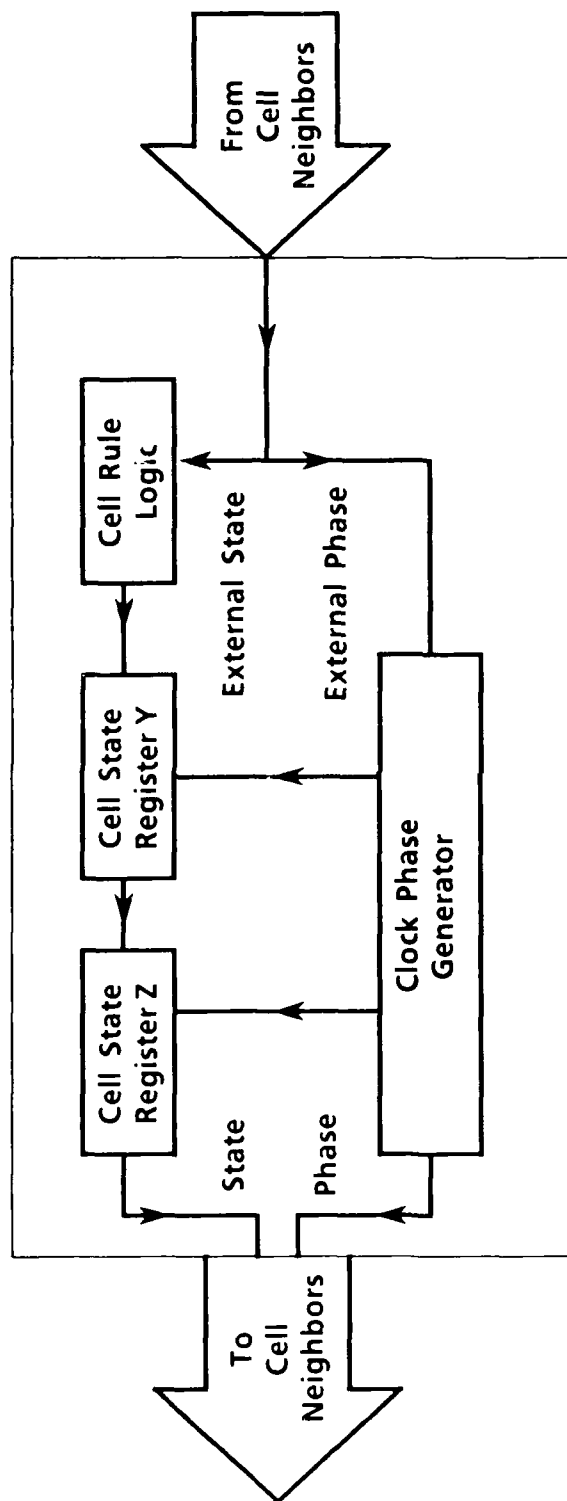


Figure 22. High-level representation of asynchronous automaton cell. The clock phase generator is an automaton that follows the interaction rule specified in Table 8.

- Phase 2: Cell X enters phase 2 when it detects that all its neighbors are in phase 1 or 2 sub-cycles. This condition implies that all local cells have performed internal process steps and no longer require the old cell state pattern. Cell X is now free to update its external state by transferring the contents of register Y to register Z. (See Figure 22.) When all neighbors have entered phase 2, all previous cell states within the neighborhood of cell X will have been updated. Cell X now automatically transits back to phase 0.

The multiphase cycle described above is allowed to repeat indefinitely. The cell interaction rule required to implement the three-phase clock operations is given in Table 8.

2.1 Experiment

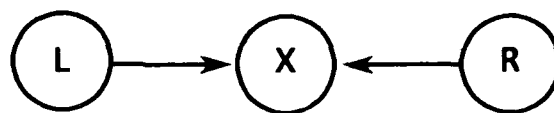
An example of the three-phase timing process for a 20-cell automaton is shown in Figure 23. Time steps are shown as new row entries in the vertical sequence. A random number generator was used to select which cell should perform a next-state calculation at each time step. No assumptions were made about the order in which a cell made state and clock phase changes. In plotting the timing pattern, we suppressed those time intervals that did not result in a change of clock phase within any cell. As shown, each cell waits until all its neighbors reach phase 1 before updating its own timing phase and signaling phase 2. Similarly, no cell transits from phase 2 to phase 0 unless both of its neighbors are in phase 2. It can be seen that intercell timing rapidly loses global synchronization, but local synchronization is maintained over time.

2.2 Discussion

The operation of the timing system described is similar to a two-pass application of a Muller C function on the cell neighborhood.²¹ A Muller C operation is equivalent to a bistable switch that has a built-in hysteresis. The switch changes output state only after all binary inputs have passed from all '0's to all '1's and vice versa. Two Muller C elements that span separately the $\{0,0,0\}$ - $\{1,1,1\}$ - and $\{1,1,1\}$ - $\{2,2,2\}$ -neighborhood phase conditions could be used to construct an equivalent self-timed CA. In essence, we have implemented a ternary, or three-level, Muller C function to

Table 8. Internal Cell Clock Phase vs Phase of Local Cell Neighborhood

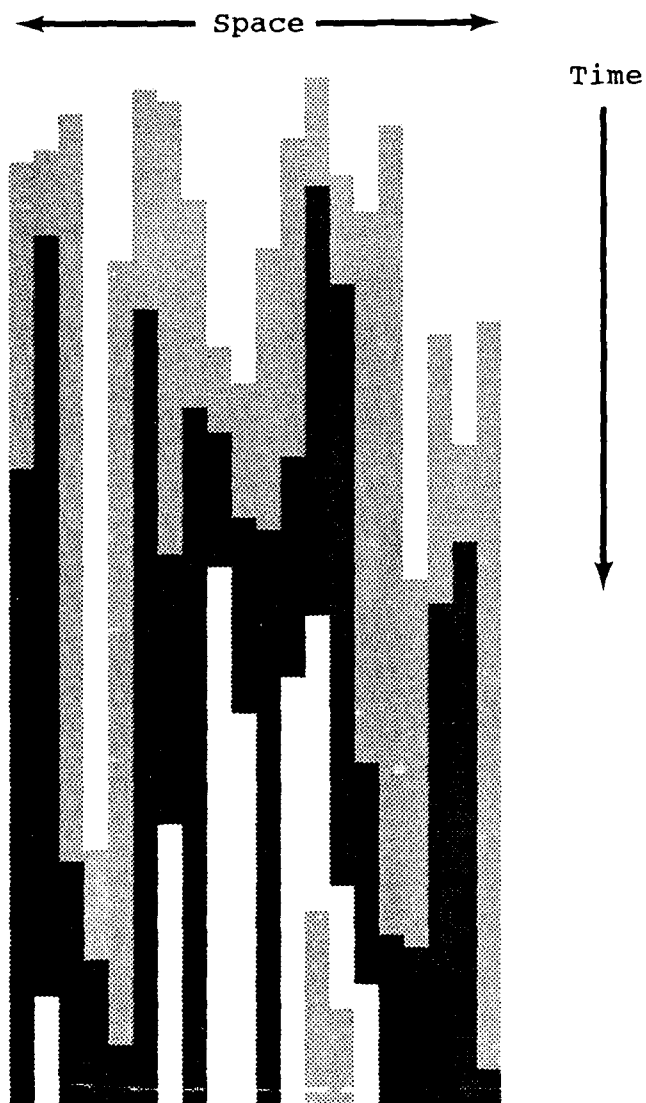
L	X	R	X'
0	0	0	1
0	0	1	1
0	0	2	0
0	1	0	1
0	1	1	1
0	1	2	1
0	2	0	0
0	2	1	2
0	2	2	0
1	0	0	1
1	0	1	1
1	0	2	0
1	1	0	1
1	1	1	2
1	1	2	2
1	2	0	2
1	2	1	2
1	2	2	2
2	0	0	0
2	0	1	0
2	0	2	0
2	1	0	1
2	1	1	2
2	1	2	2
2	2	0	0
2	2	1	2
2	2	2	0



Neighborhood of Cell X


LEGEND:

- L Left neighbor phase at time t
- X Internal cell phase at time t
- R Right neighbor phase at time t
- X' Internal cell phase at time t'



Legend

Phase 0 Cell

Phase 1 Cell 


Phase 2 Cell 

Figure 23. Time evolution of a completely self-timed cellular automaton. Shown are the clock phase generator outputs for a twenty cell, one-dimensional array.

manage intercell timing. In principle, the added cell functionality required to implement the multiphase clock can be included as an extension of any nearest-neighbor CA rule. Therefore, completely self-timed computation can be incorporated into any existing synchronous, nearest-neighbor coupled CA at the expense of additional cell complexity. Alternatively, a second CA array could be constructed to perform the timing operations and mated on a cell-by-cell basis to an existing CA network.

Initially, all cells must start in a well-defined state, since each cell carries additional internal memory. No initial phase shifts may exist between cell operations unless these internal states have been taken into account.

For simplicity, functional synchronization in the described method was confined to the neighborhood level. The effective synchronization of neighborhood intra-actions guarantees proper operation of even fully asynchronous cells. In practice, however, it might be that several cell states must be synchronized as a group. An obvious instance would be an I/O event between cell groups that must exchange large blocks of cell state values. In these cases, a more hierarchical scheme would have to be used to synchronize the interaction of cells that lie beyond the local neighborhood. This hierarchy can be implemented by adding more states to the timing graph so that the effective synchronization of cells can be expanded to synchronize entire cell neighborhoods. The solution would entail the mapping of more distant cell states into a nearest-neighbor CA, and then using the technique described here to perform the synchronization process.

REFERENCES

1. "Next-generation Computers," E. A. Torrero, ed. (IEEE Press, 1985).
2. R. T. Bate, "Limits to the Performance of VLSI Circuits," VLSI Handbook (Academic Press, 1985).
3. P. K. Chatterjee, P. Yang, and H. Shichijo, "Modeling of Small MOS Devices and Device Limits," Proc. IEEE 130 (Part 1), 3, 105 (1983).
4. C. P. Yuan and T. N. Trick, "Calculation of Capacitance in VLSI Circuits," Proc. ICCD, p. 263 (1982).
5. K. C. Saraswat and F. Mohammed, "Effect of Scaling Interconnects on Time Delay of VLSI Circuits," IEEE J. Solid-State Circuits SC-17, 275 (1982).
6. M. Uenohara, et al., "The Next Generation of Integrated Circuits - What Comes After VLSI?" in "VLSI Microstructure Science," Vol. 9, Chapter 11, p. 385 (Academic Press 1985).
7. G. A. Frazier, "Simulation of Neural Networks," International Conference on Neural Networks, Santa Barbara, CA, 1984.
8. C. A. Hamilton, et al., "A High Speed Superconducting A/D Converter," 37th Annual Device Research Conf., June 25-27 1979.
9. R. T. Bate, et al., "Prospects for Quantum Integrated Circuits," Baypoint Conference on Quantum Well and Superlattice Physics, Proc., March 1987.
10. A. C. Gossard, "Growth of Microstructures by Molecular Beam Epitaxy," IEEE J. Quantum Electronics QE-22, 9, 1649 (1986).
11. "E. Merzbacher, Quantum Mechanics (John Wiley and Sons, Inc, 1970).

12. M. A. Reed, "Excited State Resonant Tunneling in GaAs-Al_xGa_{1-x}As Double Barrier Heterostructures," Superlattices and Microstructures 2, 65 (1986).
13. W. R. Frensley, "Transient Response of a Tunneling Device Obtained From the Wigner Function," Phys. Rev. Lett. 57, 22, 2853 (1986).
14. Theory and Applications of Cellular Automata, S. Wolfram ed., (Singapore Pub., 1986).
15. E. R. Berlekamp, et al., "Winning Ways For Your Mathematics Plays," Vol. 2 (Academic Press, 1982).
16. IEEE Trans. Computers C-35, No. 2 (February 1986).
17. Z. G. Vranesic and V. C. Hamacher, "Ternary Logic in Parallel Multipliers," Journal of Comp. 15, 3, 254 (1972).
18. S. V. Jablonski, "Functional Constructions in K-valued Logics," Trudy Mat. Inst. Steklov. 51, 5-142 (1958).
19. Uenohara, Chapter 11.
20. G. A. Frazier, "An Ideology for Nanoelectronics," Princeton Workshop on Architectures, Algorithms, and Technology Issues, 1987.
21. C. Mead and L. Conway, "Introduction to VLSI Systems," Chapter 7, (Addison Wesley, 1980).

APPENDIX A

REDUCED EQUATIONS FOR NEAREST-NEIGHBOR CELLULAR AUTOMATA

APPENDIX A

REDUCED EQUATIONS FOR NEAREST-NEIGHBOR CELLULAR AUTOMATA

This appendix holds a list of all 256 interaction rules for nearest-neighbor coupled, one-dimensional cellular automata. The equations are in canonical (sum-of-products) form, and are expressed using the Boolean logic operations AND, OR, and NOT. For a given set of cell values, A, B, and C at time t , the equations predict the new state of the center cell B at time $t+1$. For example, Rule 160 requires that the next state of the center cell be the logical OR of the present state of its left and right neighbors. Rule 204 does not alter any cell states, while rule 240 simply shifts the cell pattern by one cell at each time step. These equations were examined for conjugation and reflection symmetry. We found that 88 rules are sufficient to completely account for all automaton behaviors. These rules have been marked within Table A1 by a leading asterisk and the 88 quadruplet sets of rules are listed in Table A2.

Table A1. Reduced Equations For All Nearest Neighbor, One-Dimensional Cellular Automata Rules

Rule	Equation
* 0	(0)
* 1	(A ⁻ B ⁻ C ⁻)
* 2	(A ⁻ B ⁻ C)
* 3	(A ⁻ B ⁻)
* 4	(A ⁻ BC ⁻)
* 5	(A ⁻ C ⁻)
* 6	(A ⁻ B ⁻ C)+(A ⁻ BC ⁻)
* 7	(A ⁻ B ⁻)+(A ⁻ C ⁻)
* 8	(A ⁻ BC)
* 9	(A ⁻ B ⁻ C ⁻)+(A ⁻ BC)
* 10	(A ⁻ C)
* 11	(A ⁻ B ⁻)+(A ⁻ C)
* 12	(A ⁻ B)
* 13	(A ⁻ B)+(A ⁻ C ⁻)
* 14	(A ⁻ B)+(A ⁻ C)
* 15	(A ⁻)
16	(AB ⁻ C ⁻)
17	(B ⁻ C ⁻)
* 18	(A ⁻ B ⁻ C)+(AB ⁻ C ⁻)
* 19	(A ⁻ B ⁻)+(B ⁻ C ⁻)
20	(A ⁻ BC ⁻)+(AB ⁻ C ⁻)
21	(A ⁻ C ⁻)+(B ⁻ C ⁻)
* 22	(A ⁻ B ⁻ C)+(A ⁻ BC ⁻)+(AB ⁻ C ⁻)
* 23	(A ⁻ B ⁻)+(A ⁻ C ⁻)+(B ⁻ C ⁻)
* 24	(A ⁻ BC)+(AB ⁻ C ⁻)
* 25	(B ⁻ C ⁻)+(A ⁻ B ⁻ C ⁻)+(A ⁻ BC)
* 26	(A ⁻ C)+(A ⁻ B ⁻ C)+(A ⁻ BC)+(AB ⁻ C ⁻)
* 27	(A ⁻ B ⁻)+(A ⁻ C)+(B ⁻ C ⁻)
* 28	(A ⁻ B)+(A ⁻ BC ⁻)+(A ⁻ BC)+(AB ⁻ C ⁻)
* 29	(A ⁻ B)+(A ⁻ C ⁻)+(B ⁻ C ⁻)
* 30	(A ⁻ B)+(A ⁻ C)+(A ⁻ B ⁻ C)+(A ⁻ BC ⁻)+(A ⁻ BC)+(AB ⁻ C ⁻)
31	(A ⁻)+(A ⁻ B)+(A ⁻ B ⁻)+(A ⁻ C)+(A ⁻ C ⁻)+(B ⁻ C ⁻)
* 32	(AB ⁻ C)
* 33	(A ⁻ B ⁻ C ⁻)+(AB ⁻ C)
* 34	(B ⁻ C)
* 35	(A ⁻ B ⁻)+(B ⁻ C)
* 36	(A ⁻ BC ⁻)+(AB ⁻ C)
* 37	(A ⁻ C ⁻)+(A ⁻ B ⁻ C ⁻)+(A ⁻ BC ⁻)+(AB ⁻ C)
* 38	(B ⁻ C)+(A ⁻ B ⁻ C)+(A ⁻ BC ⁻)
39	(A ⁻ B ⁻)+(A ⁻ C ⁻)+(B ⁻ C)
* 40	(A ⁻ BC)+(AB ⁻ C)

Definitions
A⁻ = NOT A
(AB) = A AND B
(A)+(B) = A OR B

Left Cell Center Cell Right Cell

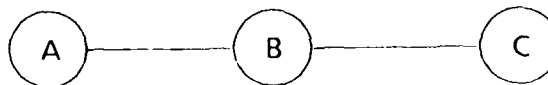


Table A1. (Continued)

Rule

Equation

*41	$(A^-B^-C^-)+(A^-BC)+(AB^-C)$
*42	$(A^-C)+(B^-C)$
*43	$(A^-B^-)+(A^-C)+(B^-C)$
*44	$(A^-B)+(A^-BC^-)+(A^-BC)+(AB^-C)$
*45	$(A^-B)+(A^-C^-)+(A^-B^-C^-)+(A^-BC^-)+(A^-BC)+(AB^-C)$
*46	$(A^-B)+(A^-C)+(B^-C)$
47	$(A^-)+(A^-B)+(A^-B^-)+(A^-C)+(A^-C^-)+(B^-C)$
48	(AB^-)
49	$(AB^-)+(B^-C^-)$
*50	$(AB^-)+(B^-C)$
*51	(B^-)
52	$(AB^-)+(A^-BC^-)$
53	$(AB^-)+(A^-C^-)$
*54	$(AB^-)+(B^-C)+(A^-B^-C)+(A^-BC^-)$
55	$(B^-)+(AB^-)+(A^-B^-)+(A^-C^-)$
*56	$(AB^-)+(A^-BC)$
*57	$(AB^-)+(B^-C^-)+(A^-B^-C^-)+(A^-BC)$
*58	$(AB^-)+(A^-C)$
59	$(B^-)+(AB^-)+(A^-B^-)+(A^-C)$
*60	$(AB^-)+(A^-B)$
61	$(AB^-)+(A^-B)+(A^-C^-)$
*62	$(AB^-)+(A^-B)+(A^-C)$
63	$(A^-)+(B^-)$
64	(ABC^-)
65	$(A^-B^-C^-)+(ABC^-)$
66	$(A^-B^-C)+(ABC^-)$
67	$(A^-B^-)+(A^-B^-C^-)+(A^-B^-C)+(ABC^-)$
68	(BC^-)
69	$(A^-C^-)+(BC^-)$
70	$(BC^-)+(A^-B^-C)$
71	$(A^-B^-)+(A^-C^-)+(BC^-)$
*72	$(A^-BC)+(ABC^-)$
*73	$(A^-B^-C^-)+(A^-BC)+(ABC^-)$
*74	$(A^-C)+(A^-B^-C)+(A^-BC)+(ABC^-)$
75	$(A^-B^-)+(A^-C)+(A^-B^-C^-)+(A^-B^-C)+(A^-BC)+(ABC^-)$
*76	$(A^-B)+(BC^-)$
*77	$(A^-B)+(A^-C^-)+(BC^-)$
*78	$(A^-B)+(A^-C)+(BC^-)$
79	$(A^-)+(A^-B)+(A^-B^-)+(A^-C)+(A^-C^-)+(BC^-)$
80	(AC^-)

Definitions

A^- = NOT A
 (AB) = A AND B
 $(A)+(B)$ = A OR B

Left Cell Center Cell Right Cell

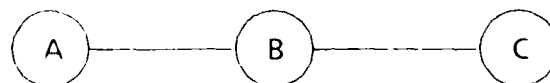


Table A1. (Continued)

Rule	Equation
81	$(AC^-) + (B^-C^-)$
82	$(AC^-) + (A^-B^-C)$
83	$(AC^-) + (A^-B^-)$
84	$(AC^-) + (BC^-)$
85	(C^-)
86	$(AC^-) + (BC^-) + (A^-B^-C)$
87	$(C^-) + (AC^-) + (A^-B^-)$
88	$(AC^-) + (A^-BC)$
89	$(AC^-) + (B^-C^-) + (A^-B^-C^-) + (A^-BC)$
*90	$(AC^-) + (A^-C)$
91	$(AC^-) + (A^-B^-) + (A^-C)$
92	$(AC^-) + (A^-B)$
93	$(C^-) + (AC^-) + (A^-B)$
*94	$(AC^-) + (A^-B) + (A^-C)$
95	$(A^-) + (C^-)$
96	$(AB^-C) + (ABC^-)$
97	$(A^-B^-C^-) + (AB^-C) + (ABC^-)$
98	$(B^-C) + (A^-B^-C) + (AB^-C) + (ABC^-)$
99	$(A^-B^-) + (B^-C) + (A^-B^-C^-) + (A^-B^-C) + (AB^-C) + (ABC^-)$
100	$(BC^-) + (A^-BC^-) + (AB^-C)$
101	$(A^-C^-) + (BC^-) + (A^-B^-C^-) + (A^-BC^-) + (AB^-C)$
102	$(BC^-) + (B^-C)$
103	$(A^-B^-) + (A^-C^-) + (BC^-) + (B^-C)$
*104	$(A^-BC) + (AB^-C) + (ABC^-)$
*105	$(A^-B^-C^-) + (A^-BC) + (AB^-C) + (ABC^-)$
*106	$(A^-C) + (B^-C) + (A^-B^-C) + (A^-BC) + (AB^-C) + (ABC^-)$
107	$(A^-B^-) + (A^-C) + (B^-C) + (A^-B^-C^-) + (A^-B^-C) + (A^-BC) + (AB^-C) + (ABC^-)$
*108	$(A^-B) + (BC^-) + (A^-BC^-) + (A^-BC) + (AB^-C)$
109	$(A^-B) + (A^-C^-) + (BC^-) + (A^-B^-C^-) + (A^-BC^-) + (A^-BC) + (AB^-C)$
*110	$(A^-B) + (A^-C) + (BC^-) + (B^-C)$
111	$(A^-) + (A^-B) + (A^-B^-) + (A^-C) + (A^-C^-) + (BC^-) + (B^-C)$
112	$(AB^-) + (AC^-)$
113	$(AB^-) + (AC^-) + (B^-C^-)$
114	$(AB^-) + (AC^-) + (B^-C)$
115	$(B^-) + (AB^-) + (AC^-)$
116	$(AB^-) + (AC^-) + (BC^-)$
117	$(C^-) + (AB^-)$
118	$(AB^-) + (AC^-) + (BC^-) + (B^-C)$
119	$(B^-) + (C^-)$
120	$(AB^-) + (AC^-) + (A^-BC)$

Definitions

 A^- = NOT A (AB) = A AND B $(A) + (B)$ = A OR B

Left Cell Center Cell Right Cell

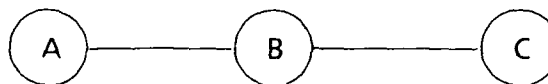


Table A1. (Continued)

Rule	Equation
121	$(AB^-) + (AC^-) + (B^-C^-) + (A^-B^-C^-) + (A^-BC)$
*122	$(AB^-) + (AC^-) + (A^-C)$
123	$(B^-) + (AB^-) + (AC^-) + (A^-B^-) + (A^-C)$
124	$(AB^-) + (AC^-) + (A^-B)$
125	$(C^-) + (AB^-) + (AC^-) + (A^-B)$
*126	$(AB^-) + (AC^-) + (A^-B) + (A^-C)$
127	$(A^-) + (B^-) + (C^-)$
*128	(ABC)
129	$(A^-B^-C^-) + (ABC)$
*130	$(A^-B^-C) + (ABC)$
131	$(A^-B^-) + (A^-B^-C^-) + (A^-B^-C) + (ABC)$
132	$(A^-BC^-) + (ABC)$
133	$(A^-C^-) + (A^-B^-C^-) + (A^-BC^-) + (ABC)$
*134	$(A^-B^-C) + (A^-BC^-) + (ABC)$
135	$(A^-B^-) + (A^-C^-) + (A^-B^-C^-) + (A^-B^-C) + (A^-BC^-) + (ABC)$
*136	(BC)
137	$(BC) + (A^-B^-C^-)$
*138	$(A^-C) + (BC)$
139	$(A^-B^-) + (A^-C) + (BC)$
*140	$(A^-B) + (BC)$
141	$(A^-B) + (A^-C^-) + (BC)$
*142	$(A^-B) + (A^-C) + (BC)$
143	$(A^-) + (A^-B) + (A^-B^-) + (A^-C) + (A^-C^-) + (BC)$
144	$(AB^-C^-) + (ABC)$
145	$(B^-C^-) + (A^-B^-C^-) + (AB^-C^-) + (ABC)$
*146	$(A^-B^-C) + (AB^-C^-) + (ABC)$
147	$(A^-B^-) + (B^-C^-) + (A^-B^-C^-) + (A^-B^-C) + (AB^-C^-) + (ABC)$
148	$(A^-BC^-) + (AB^-C^-) + (ABC)$
149	$(A^-C^-) + (B^-C^-) + (A^-B^-C^-) + (A^-BC^-) + (AB^-C^-) + (ABC)$
*150	$(A^-B^-C) + (A^-BC^-) + (AB^-C^-) + (ABC)$
151	$(A^-B^-) + (A^-C^-) + (B^-C^-) + (A^-B^-C^-) + (A^-B^-C) + (A^-BC^-) + (AB^-C^-) + (ABC)$
*152	$(BC) + (A^-BC) + (AB^-C^-)$
153	$(BC) + (B^-C^-)$
*154	$(A^-C) + (BC) + (A^-B^-C) + (A^-BC) + (AB^-C^-)$
155	$(A^-B^-) + (A^-C) + (BC) + (B^-C^-)$
*156	$(A^-B) + (BC) + (A^-BC^-) + (A^-BC) + (AB^-C^-)$
157	$(A^-B) + (A^-C^-) + (BC) + (B^-C^-)$
158	$(A^-B) + (A^-C) + (BC) + (A^-B^-C) + (A^-BC^-) + (A^-BC) + (AB^-C^-)$
159	$(A^-) + (A^-B) + (A^-B^-) + (A^-C) + (A^-C^-) + (BC) + (B^-C^-)$
*160	(AC)

Definitions

 $A^- = \text{NOT } A$ $(AB) = A \text{ AND } B$ $(A) + (B) = A \text{ OR } B$

Left Cell Center Cell Right Cell

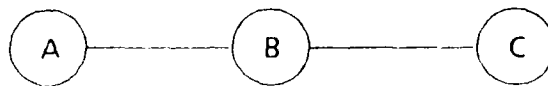


Table A1. (Continued)

Rule

Equation

161	$(AC) + (A^{\sim}B^{\sim}C^{\sim})$
*162	$(AC) + (B^{\sim}C)$
163	$(AC) + (A^{\sim}B^{\sim})$
*164	$(AC) + (A^{\sim}BC^{\sim})$
165	$(AC) + (A^{\sim}C^{\sim})$
166	$(AC) + (B^{\sim}C) + (A^{\sim}B^{\sim}C) + (A^{\sim}BC^{\sim})$
167	$(AC) + (A^{\sim}B^{\sim}) + (A^{\sim}C^{\sim})$
*168	$(AC) + (BC)$
169	$(AC) + (BC) + (A^{\sim}B^{\sim}C^{\sim})$
*170	(C)
171	$(C) + (AC) + (A^{\sim}B^{\sim})$
*172	$(AC) + (A^{\sim}B)$
173	$(AC) + (A^{\sim}B) + (A^{\sim}C^{\sim})$
174	$(C) + (AC) + (A^{\sim}B)$
175	$(A^{\sim}) + (C)$
176	$(AB^{\sim}) + (AC)$
177	$(AB^{\sim}) + (AC) + (B^{\sim}C^{\sim})$
*178	$(AB^{\sim}) + (AC) + (B^{\sim}C)$
179	$(B^{\sim}) + (AB^{\sim}) + (AC)$
180	$(AB^{\sim}) + (AC) + (A^{\sim}BC^{\sim})$
181	$(AB^{\sim}) + (AC) + (A^{\sim}C^{\sim})$
182	$(AB^{\sim}) + (AC) + (B^{\sim}C) + (A^{\sim}B^{\sim}C) + (A^{\sim}BC^{\sim})$
183	$(B^{\sim}) + (AB^{\sim}) + (AC) + (A^{\sim}B^{\sim}) + (A^{\sim}C^{\sim})$
*184	$(AB^{\sim}) + (AC) + (BC)$
185	$(AB^{\sim}) + (AC) + (BC) + (B^{\sim}C^{\sim})$
186	$(C) + (AB^{\sim})$
187	$(B^{\sim}) + (C)$
188	$(AB^{\sim}) + (AC) + (A^{\sim}B)$
189	$(AB^{\sim}) + (AC) + (A^{\sim}B) + (A^{\sim}C^{\sim})$
190	$(C) + (AB^{\sim}) + (AC) + (A^{\sim}B)$
191	$(A^{\sim}) + (B^{\sim}) + (C)$
192	(AB)
193	$(AB) + (A^{\sim}B^{\sim}C^{\sim})$
194	$(AB) + (A^{\sim}B^{\sim}C)$
195	$(AB) + (A^{\sim}B^{\sim})$
196	$(AB) + (BC^{\sim})$
197	$(AB) + (A^{\sim}C^{\sim})$
198	$(AB) + (BC^{\sim}) + (A^{\sim}B^{\sim}C)$
199	$(AB) + (A^{\sim}B^{\sim}) + (A^{\sim}C^{\sim})$
*200	$(AB) + (BC)$

Definitions

 $A^{\sim} = \text{NOT } A$ $(AB) = A \text{ AND } B$ $(A)+(B) = A \text{ OR } B$

Left Cell

Center Cell

Right Cell

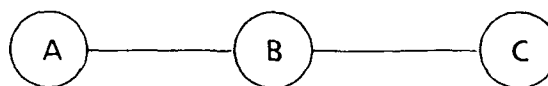


Table A1. (Continued)
Equation

Rule

201	$(AB) + (BC) + (A^-B^-C^-)$
202	$(AB) + (A^-C)$
203	$(AB) + (A^-B^-) + (A^-C)$
*204	(B)
205	$(B) + (AB) + (A^-B) + (A^-C^-)$
206	$(B) + (AB) + (A^-B) + (A^-C)$
207	$(A^-) + (B)$
208	$(AB) + (AC^-)$
209	$(AB) + (AC^-) + (B^-C^-)$
210	$(AB) + (AC^-) + (A^-B^-C)$
211	$(AB) + (AC^-) + (A^-B^-)$
212	$(AB) + (AC^-) + (BC^-)$
213	$(C^-) + (AB)$
214	$(AB) + (AC^-) + (BC^-) + (A^-B^-C)$
215	$(C^-) + (AB) + (AC^-) + (A^-B^-)$
216	$(AB) + (AC^-) + (BC)$
217	$(AB) + (AC^-) + (BC) + (B^-C^-)$
218	$(AB) + (AC^-) + (A^-C)$
219	$(AB) + (AC^-) + (A^-B^-) + (A^-C)$
220	$(B) + (AB) + (AC^-)$
221	$(B) + (C^-)$
222	$(B) + (AB) + (AC^-) + (A^-B) + (A^-C)$
223	$(A^-) + (B) + (C^-)$
224	$(AB) + (AC)$
225	$(AB) + (AC) + (A^-B^-C^-)$
226	$(AB) + (AC) + (B^-C)$
227	$(AB) + (AC) + (A^-B^-)$
228	$(AB) + (AC) + (BC^-)$
229	$(AB) + (AC) + (A^-C^-)$
230	$(AB) + (AC) + (BC^-) + (B^-C)$
231	$(AB) + (AC) + (A^-B^-) + (A^-C^-)$
*232	$(AB) + (AC) + (BC)$
233	$(AB) + (AC) + (BC) + (A^-B^-C^-)$
234	$(C) + (AB)$
235	$(C) + (AB) + (AC) + (A^-B^-)$
236	$(B) + (AB) + (AC)$
237	$(B) + (AB) + (AC) + (A^-B) + (A^-C^-)$
238	$(B) + (C)$
239	$(A^-) + (B) + (C)$
240	(A)

Definitions

A^- = NOT A
 (AB) = A AND B
 $(A) + (B)$ = A OR B

Left Cell Center Cell Right Cell

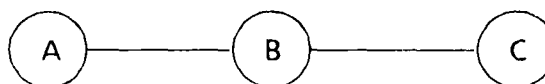


Table A1. (Continued)

Rule	Equation
241	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (B^{\sim}C^{\sim})$
242	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (B^{\sim}C)$
243	$(A) + (B^{\sim})$
244	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (BC^{\sim})$
245	$(A) + (C^{\sim})$
246	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (BC^{\sim}) + (B^{\sim}C)$
247	$(A) + (B^{\sim}) + (C^{\sim})$
248	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (BC)$
249	$(A) + (AB) + (AB^{\sim}) + (AC) + (AC^{\sim}) + (BC) + (B^{\sim}C^{\sim})$
250	$(A) + (C)$
251	$(A) + (B^{\sim}) + (C)$
252	$(A) + (B)$
253	$(A) + (B) + (C^{\sim})$
254	$(A) + (B) + (C)$
255	(1)

Definitions

A^{\sim} = NOT A
 (AB) = A AND B
 $(A) + (B)$ = A OR B

Left Cell Center Cell Right Cell

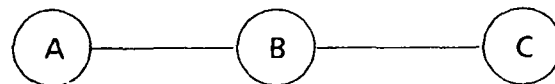


Table A2. Quadruplets of conjugation- and reflection-symmetric rules for nearest-neighbor, coupled one dimensional automata.

Quad Number	Rule Set			
1	0	0	255	255
2	1	1	127	127
3	2	16	191	247
4	3	17	63	119
5	4	4	223	223
6	5	5	95	95
7	6	20	159	215
8	7	21	31	87
9	8	64	239	253
10	9	65	111	125
11	10	80	175	245
12	11	81	47	117
13	12	68	207	221
14	13	69	79	93
15	14	84	143	213
16	15	85	15	85
17	18	18	183	183
18	19	19	55	55
19	22	22	151	151
20	23	23	23	23
21	24	66	231	189
22	25	67	103	61
23	26	82	167	181
24	27	83	39	53
25	28	70	199	157
26	29	71	71	29
27	30	86	135	149
28	32	32	251	251
29	33	33	123	123
30	34	48	187	243

Table A2. (continued)

Quad Number	Rule Set			
31	35	49	59	115
32	36	36	219	219
33	37	37	91	91
34	38	52	155	211
35	40	96	235	249
36	41	97	107	121
37	42	112	171	241
38	43	113	43	113
39	44	100	203	217
40	45	101	75	89
41	46	116	139	209
42	50	50	179	179
43	51	51	51	51
44	54	54	147	147
45	56	98	227	185
46	57	99	99	57
47	58	114	163	177
48	62	118	131	145
50	72	72	237	237
51	73	73	109	109
52	74	88	173	229
53	76	76	205	205
54	77	77	77	77
55	78	92	141	197
56	90	90	165	165
57	94	94	133	133
58	104	104	233	233
59	105	105	105	105
60	106	120	169	225

Table A2. (continued)

Quad Number	Rule Set			
61	108	108	201	201
62	110	124	137	193
63	122	122	161	161
64	126	126	129	129
65	128	128	254	254
66	130	144	190	246
67	132	132	222	222
68	134	148	158	214
69	136	192	238	252
70	138	208	174	244
71	140	196	206	220
72	142	212	142	212
73	146	146	182	182
74	150	150	150	150
75	152	194	230	188
76	154	210	166	180
77	156	198	198	156
78	160	160	250	250
79	162	176	186	242
80	164	164	218	218
81	168	224	234	248
82	170	240	170	240
83	172	228	202	216
84	178	178	178	178
85	184	226	226	184
86	200	200	236	236
87	204	204	204	204
88	232	232	232	232

APPENDIX B

SPACE-TIME PLOTS OF NEAREST-NEIGHBOR CELLULAR AUTOMATA

APPENDIX B

SPACE-TIME PLOTS OF NEAREST-NEIGHBOR CELLULAR AUTOMATA

In this appendix, we include graphical representations of the time evolution of a twenty eight (28) cell, homogeneous, one-dimensional automaton. The cell states at each time step are shown horizontally while the changes in cell state are plotted in a vertical line. The randomly selected starting state for each simulation is shown as the isolated pattern of cell values at the top of each plot. Cells in state "1" and "0" are shown in dark and light shading respectively.

The ends of the linear array of cells were coupled together so that the cells formed a closed loop. This explains why rules such as 6,31, and 151 appear to wrap around the edges of the plots. The edge cells are actually nearest neighbors in these simulations. Of the 256 rules shown, only 88 rule operations are independent. For example, rules 14,84,143, and 213 lead to identical dynamics after reflection and/or conjugation of the observed patterns. Refer to Appendix A for a list of the unique rules and their mathematical representation. .

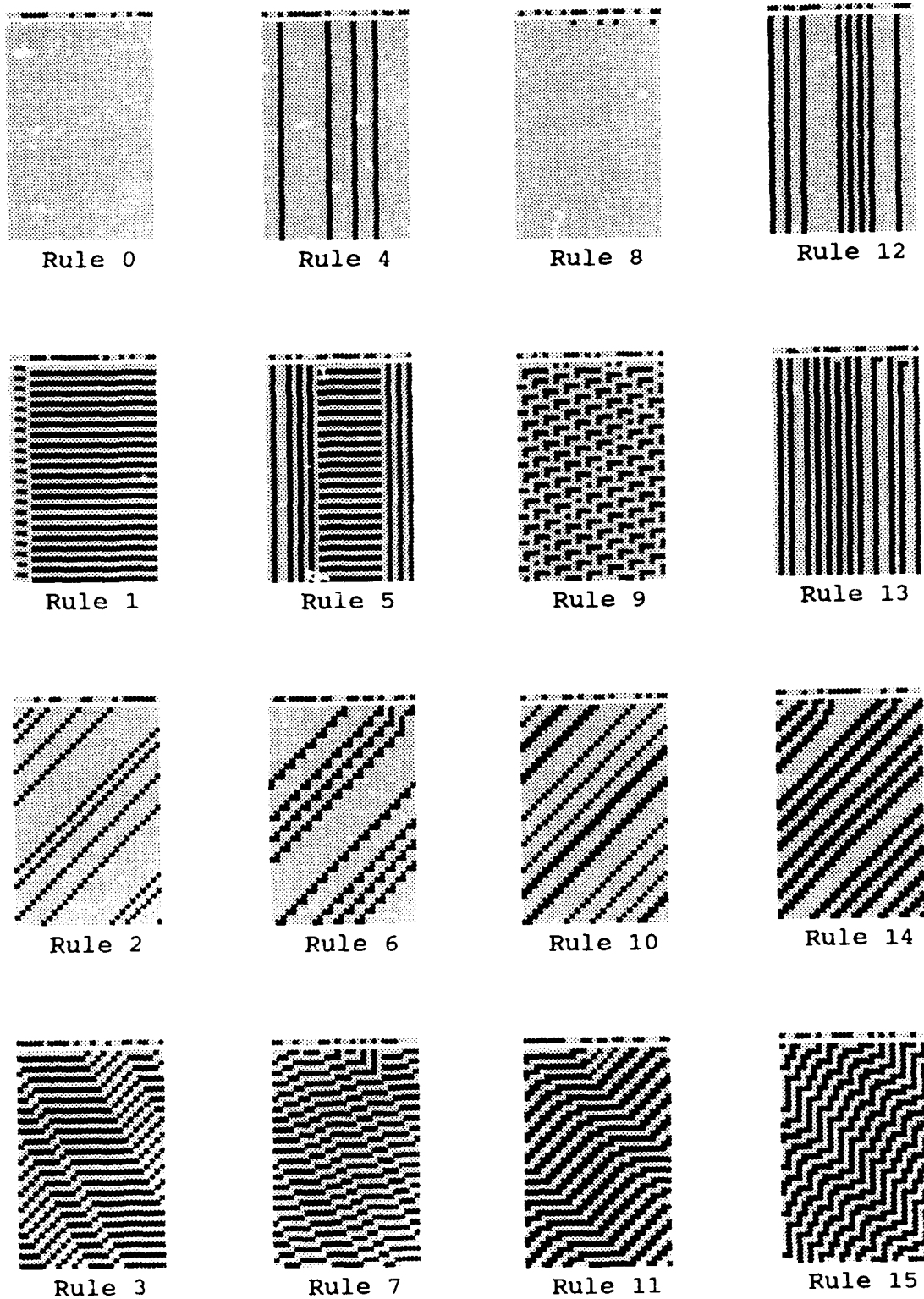
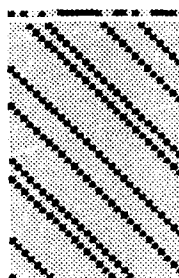
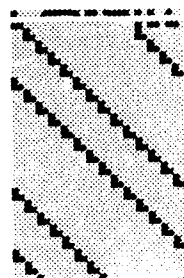


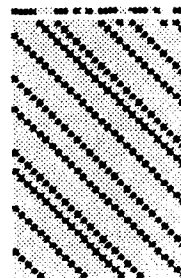
Figure B1. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



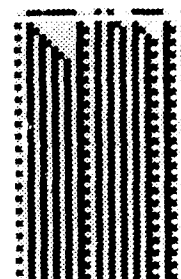
Rule 16



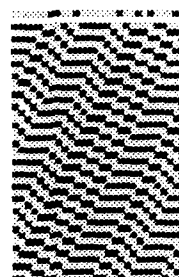
Rule 20



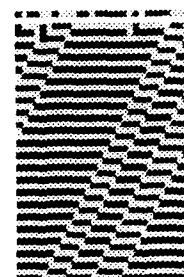
Rule 24



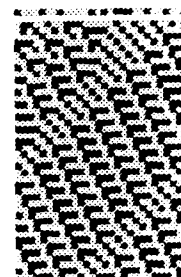
Rule 28



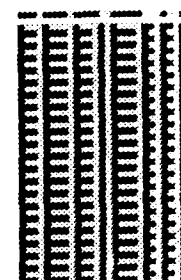
Rule 17



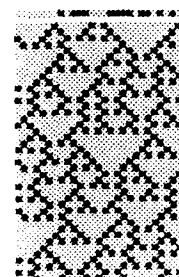
Rule 21



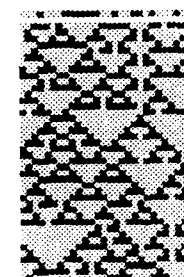
Rule 25



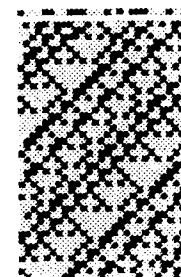
Rule 29



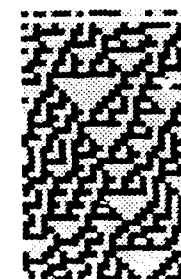
Rule 18



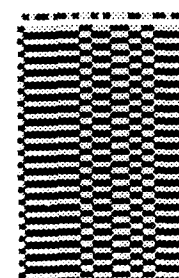
Rule 22



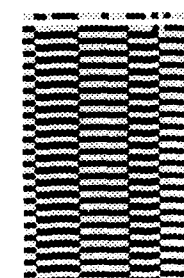
Rule 26



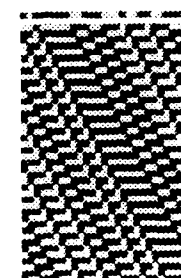
Rule 30



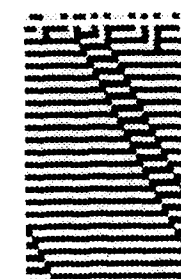
Rule 19



Rule 23

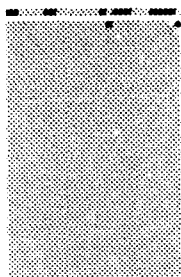


Rule 27

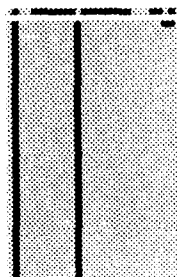


Rule 31

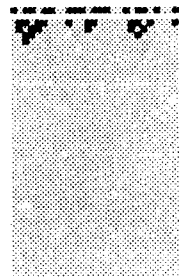
Figure B2. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



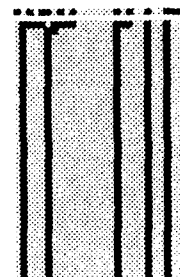
Rule 32



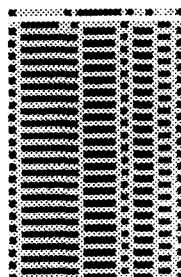
Rule 36



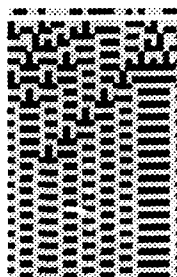
Rule 40



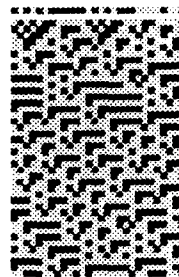
Rule 44



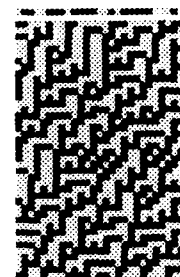
Rule 33



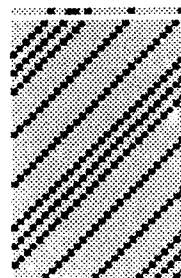
Rule 37



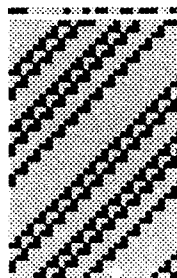
Rule 41



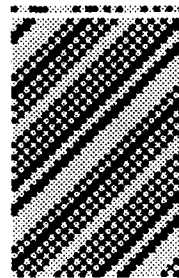
Rule 45



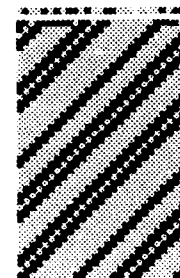
Rule 34



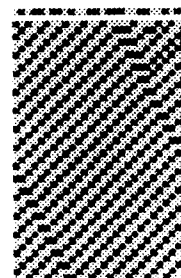
Rule 38



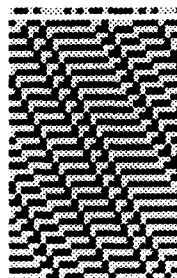
Rule 42



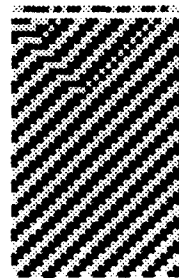
Rule 46



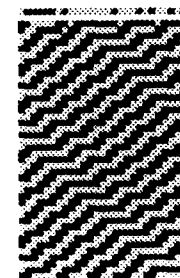
Rule 35



Rule 39

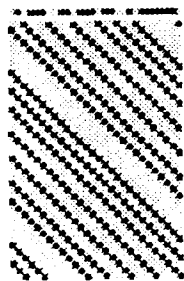


Rule 43

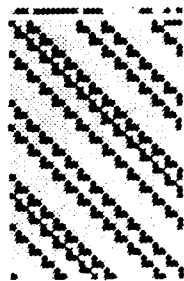


Rule 47

Figure B3. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



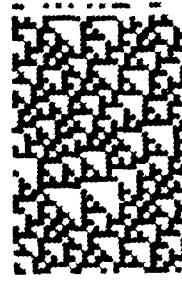
Rule 48



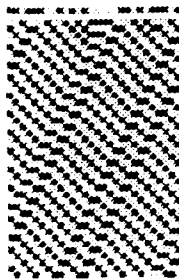
Rule 52



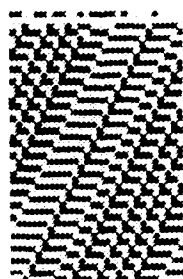
Rule 56



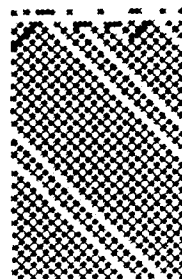
Rule 60



Rule 49



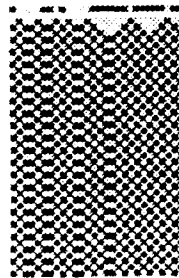
Rule 53



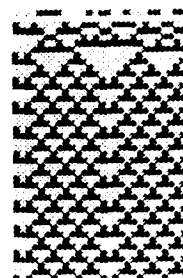
Rule 57



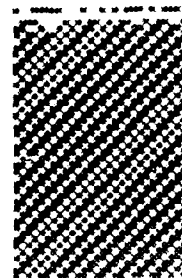
Rule 61



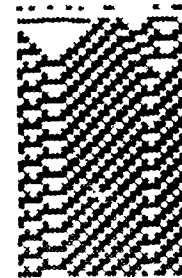
Rule 50



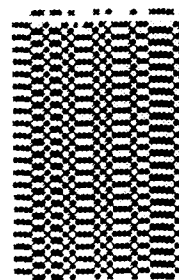
Rule 54



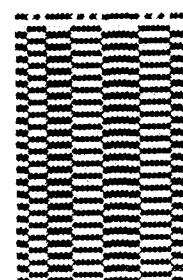
Rule 58



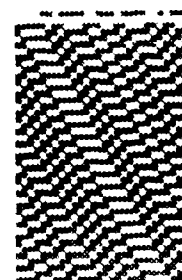
Rule 62



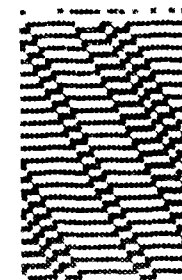
Rule 51



Rule 55



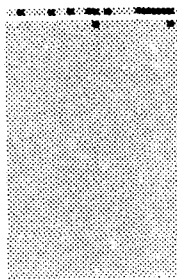
Rule 59



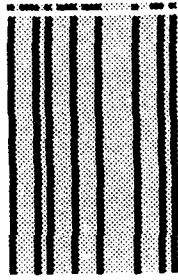
Rule 63

Figure B4.

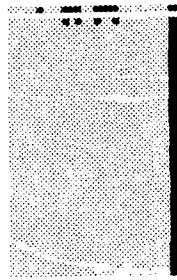
Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



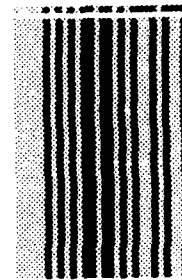
Rule 64



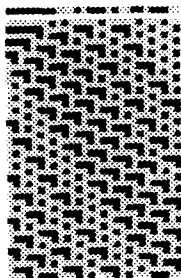
Rule 68



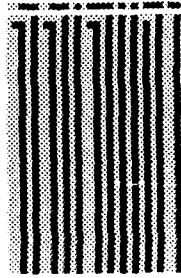
Rule 72



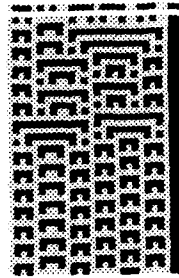
Rule 76



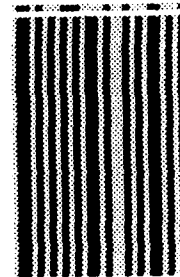
Rule 65



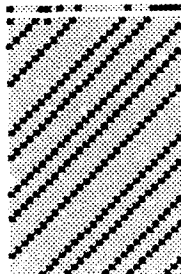
Rule 69



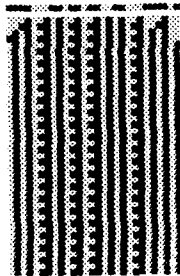
Rule 73



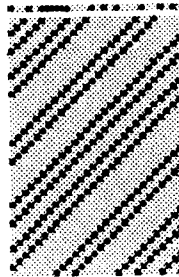
Rule 77



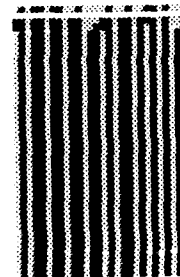
Rule 66



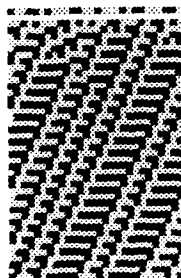
Rule 70



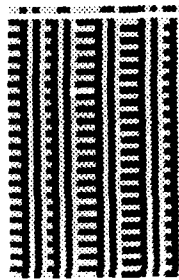
Rule 74



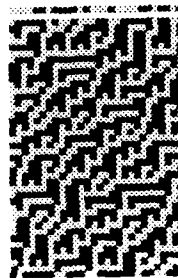
Rule 78



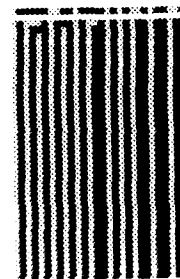
Rule 67



Rule 71

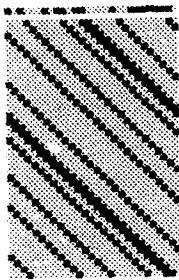


Rule 75

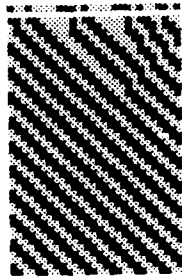


Rule 79

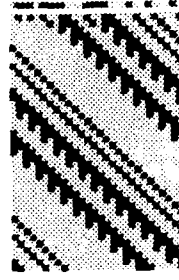
Figure B5. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



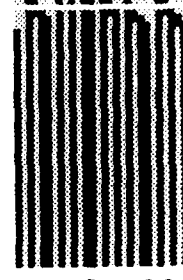
Rule 80



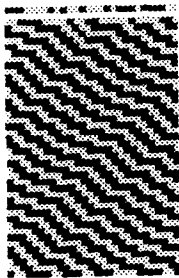
Rule 84



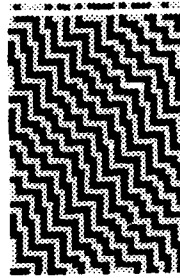
Rule 88



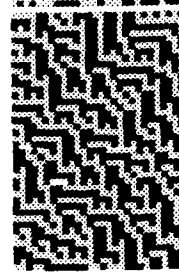
Rule 92



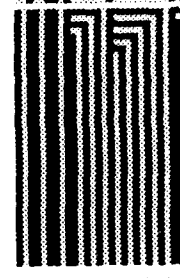
Rule 81



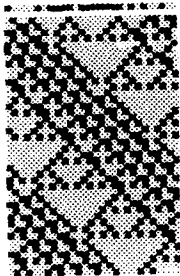
Rule 85



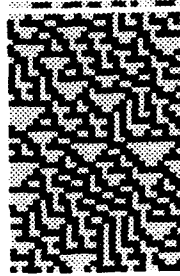
Rule 89



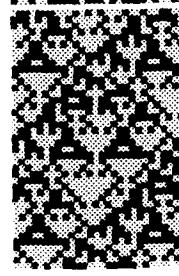
Rule 93



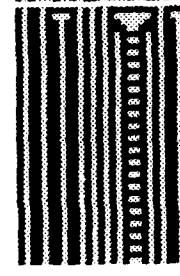
Rule 82



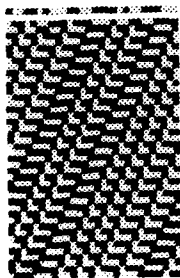
Rule 86



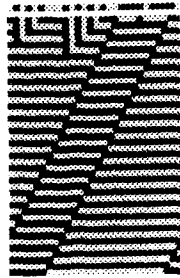
Rule 90



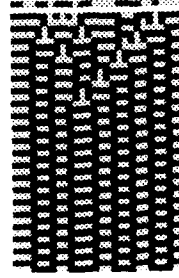
Rule 94



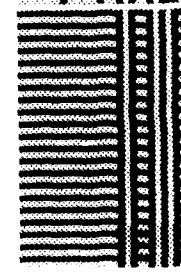
Rule 83



Rule 87

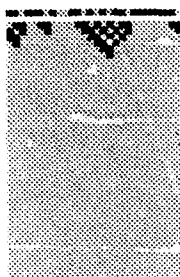


Rule 91



Rule 95

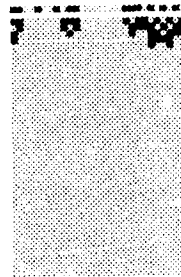
Figure B6. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



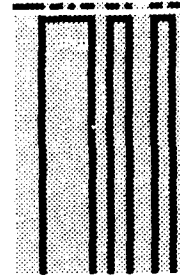
Rule 96



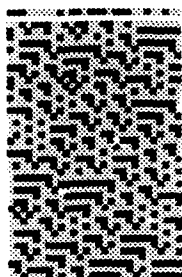
Rule 100



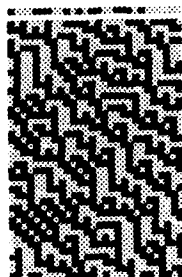
Rule 104



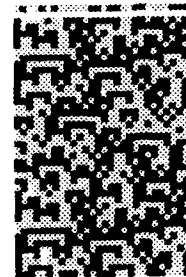
Rule 108



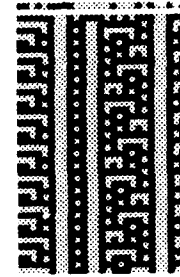
Rule 97



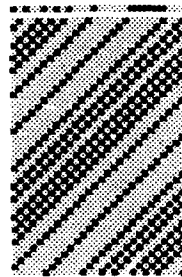
Rule 101



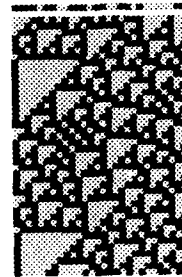
Rule 105



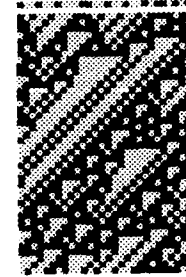
Rule 109



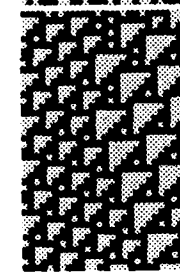
Rule 98



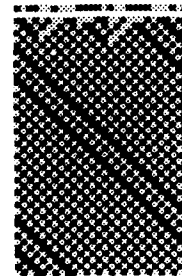
Rule 102



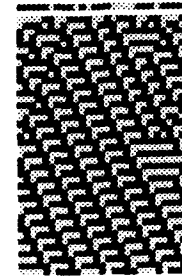
Rule 106



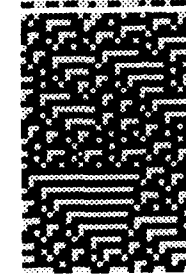
Rule 110



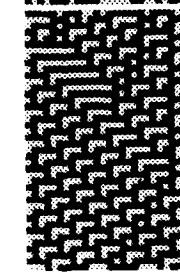
Rule 99



Rule 103

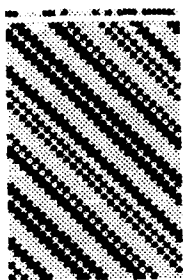


Rule 107

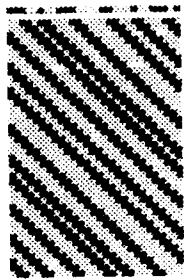


Rule 111

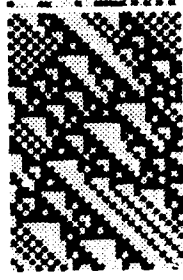
Figure B7. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



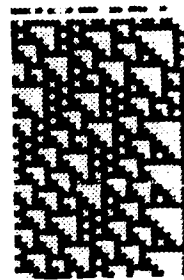
Rule 112



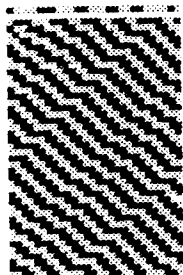
Rule 116



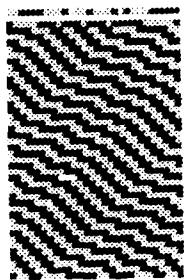
Rule 120



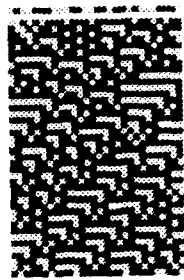
Rule 124



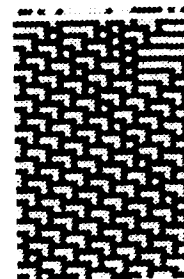
Rule 113



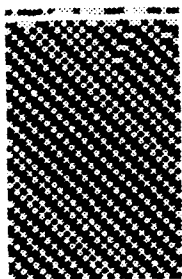
Rule 117



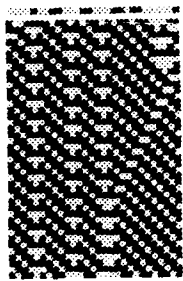
Rule 121



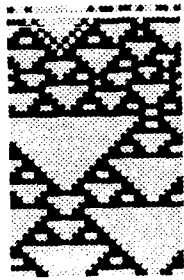
Rule 125



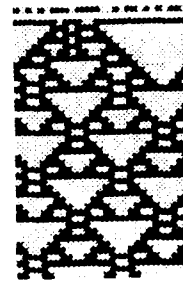
Rule 114



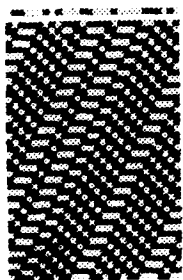
Rule 118



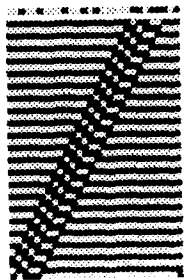
Rule 122



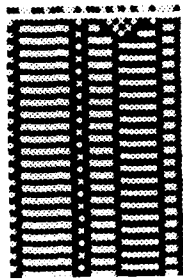
Rule 126



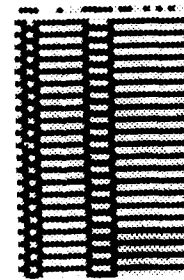
Rule 115



Rule 119



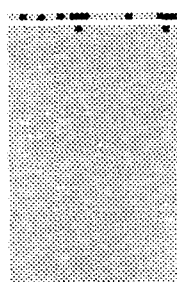
Rule 123



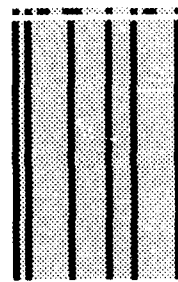
Rule 127

Figure B8.

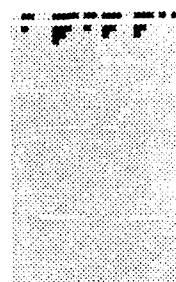
Space-time plots of nearest-neighbor cellular automata.
Rule is defined in Appendix A.



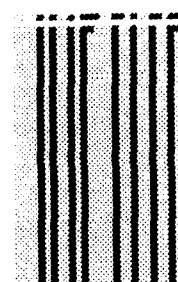
Rule 128



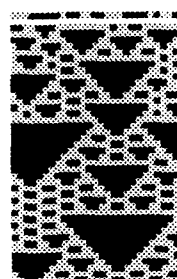
Rule 132



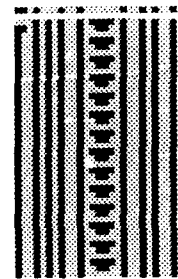
Rule 136



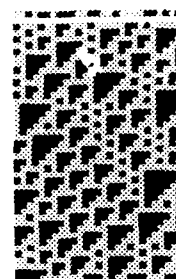
Rule 140



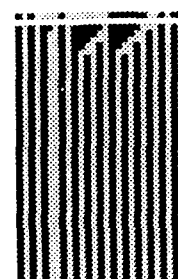
Rule 129



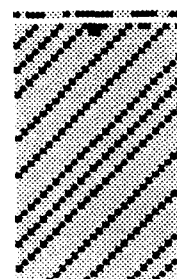
Rule 133



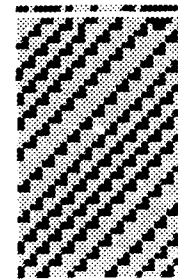
Rule 137



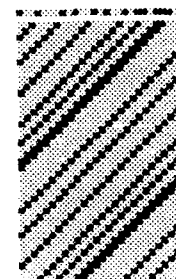
Rule 141



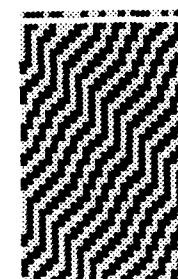
Rule 130



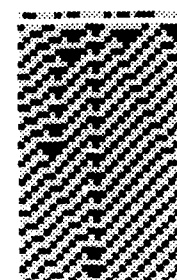
Rule 134



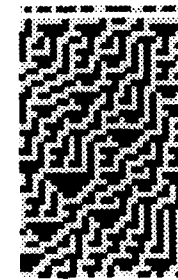
Rule 138



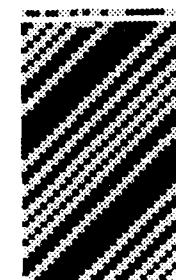
Rule 142



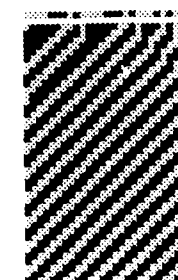
Rule 131



Rule 135

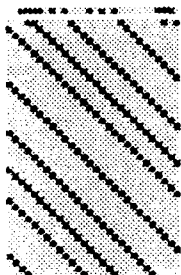


Rule 139

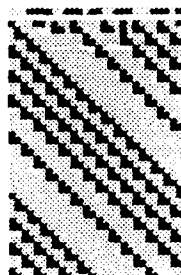


Rule 143

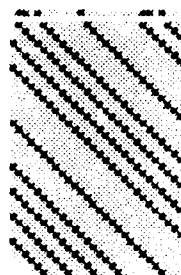
Figure 89. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



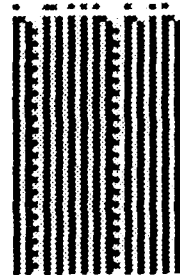
Rule 144



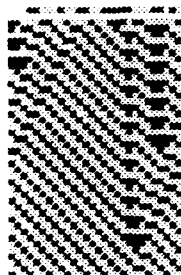
Rule 148



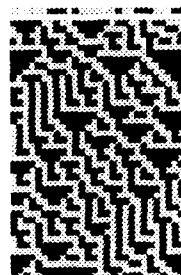
Rule 152



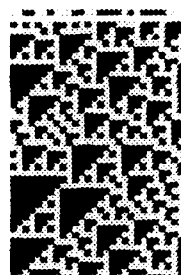
Rule 156



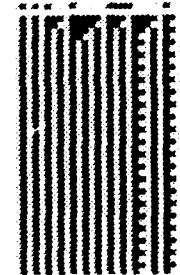
Rule 145



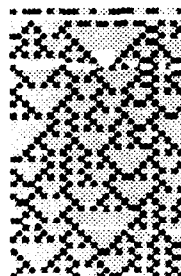
Rule 149



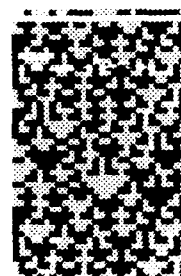
Rule 153



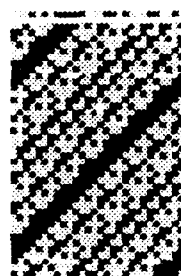
Rule 157



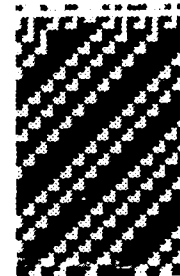
Rule 146



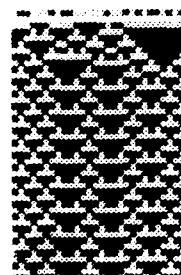
Rule 150



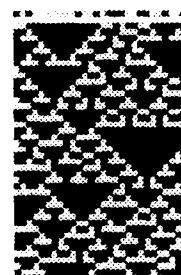
Rule 154



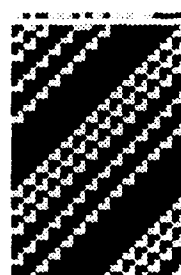
Rule 158



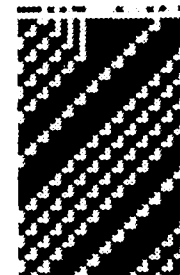
Rule 147



Rule 151

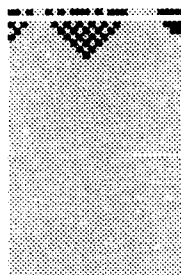


Rule 155

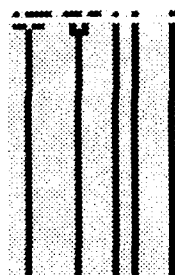


Rule 159

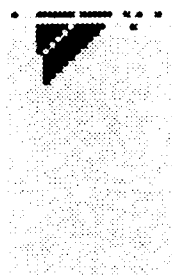
Figure B10. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



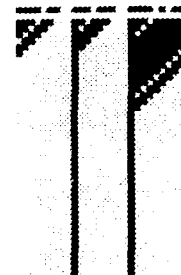
Rule 160



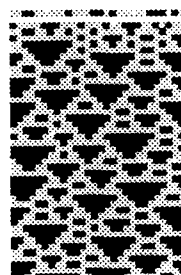
Rule 164



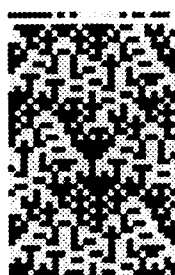
Rule 168



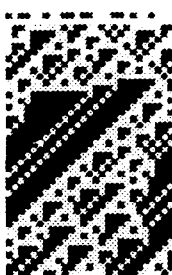
Rule 172



Rule 161



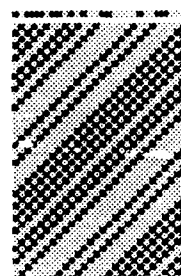
Rule 165



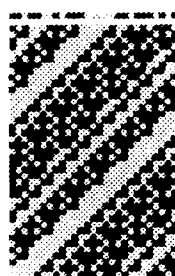
Rule 169



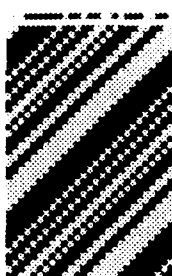
Rule 173



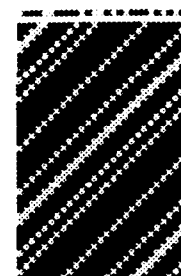
Rule 162



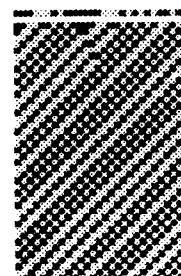
Rule 166



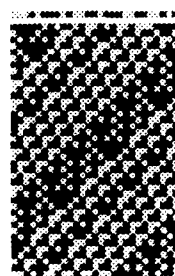
Rule 170



Rule 174



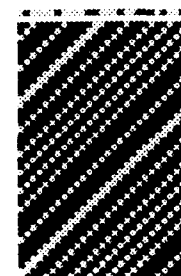
Rule 163



Rule 167

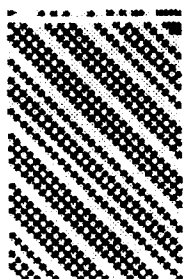


Rule 171

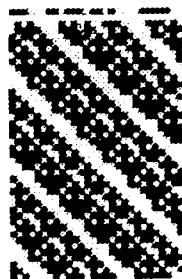


Rule 175

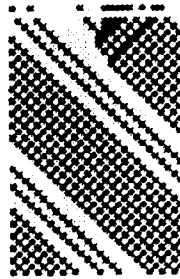
Figure B11. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



Rule 176



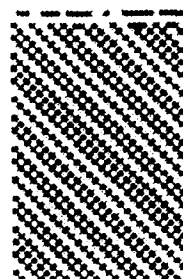
Rule 180



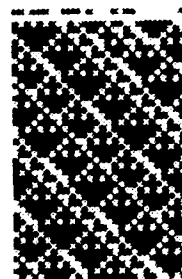
Rule 184



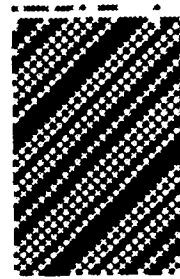
Rule 188



Rule 177



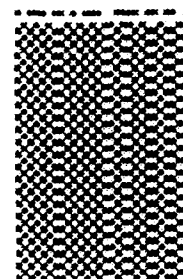
Rule 181



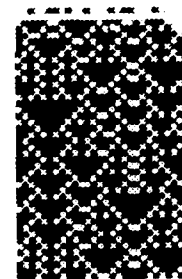
Rule 185



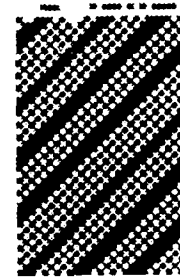
Rule 189



Rule 178



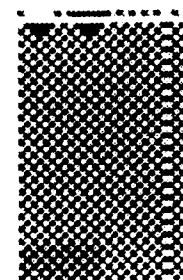
Rule 182



Rule 186



Rule 190



Rule 179



Rule 183

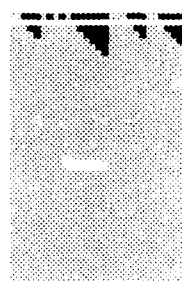


Rule 187

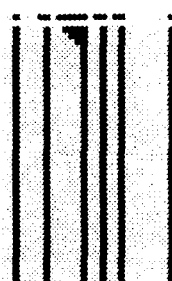


Rule 191

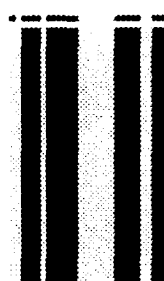
Figure B12. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



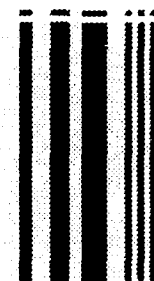
Rule 192



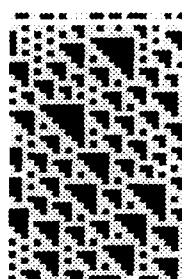
Rule 196



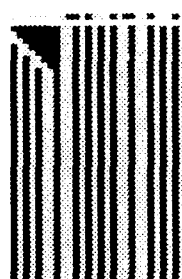
Rule 200



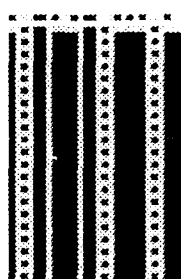
Rule 204



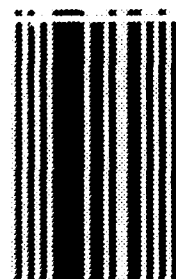
Rule 193



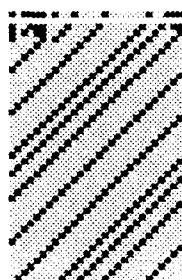
Rule 197



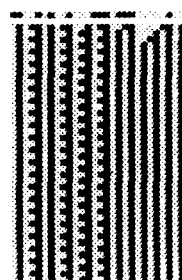
Rule 201



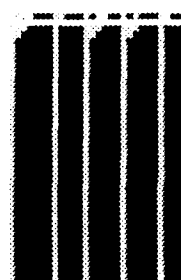
Rule 205



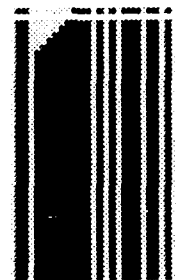
Rule 194



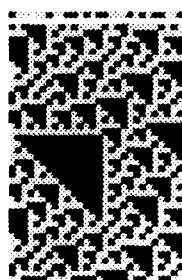
Rule 198



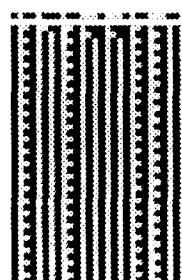
Rule 202



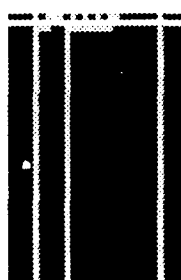
Rule 206



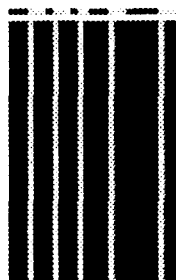
Rule 195



Rule 199

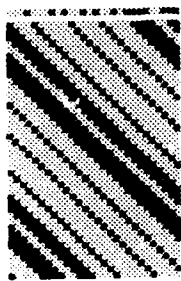


Rule 203

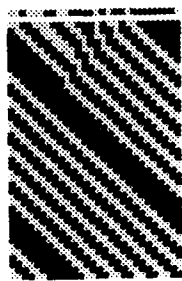


Rule 207

Figure B13. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



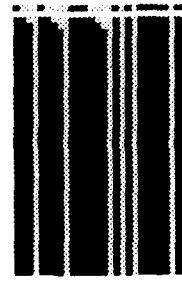
Rule 208



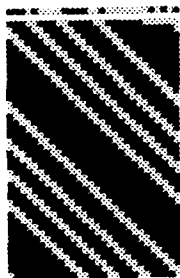
Rule 212



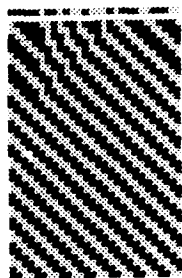
Rule 216



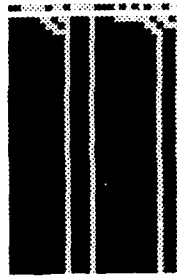
Rule 220



Rule 209



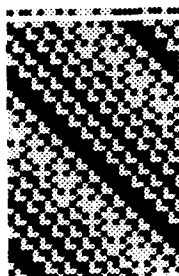
Rule 213



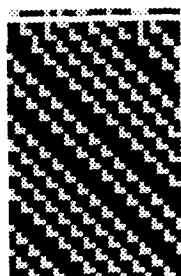
Rule 217



Rule 221



Rule 210



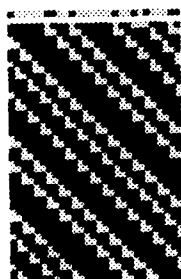
Rule 214



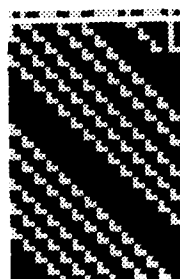
Rule 218



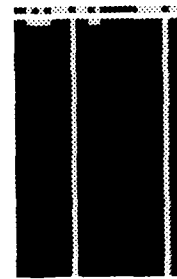
Rule 222



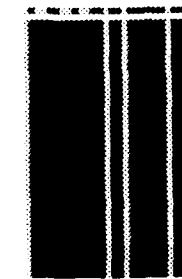
Rule 211



Rule 215

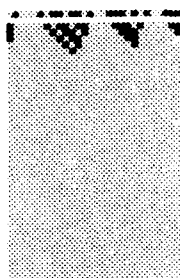


Rule 219

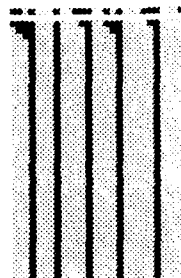


Rule 223

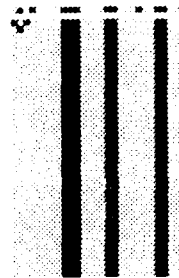
Figure B14. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



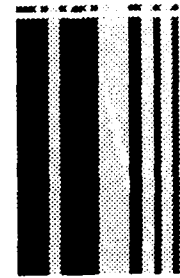
Rule 224



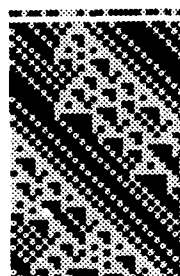
Rule 228



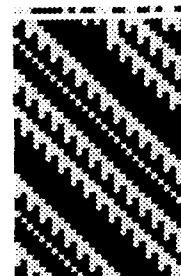
Rule 232



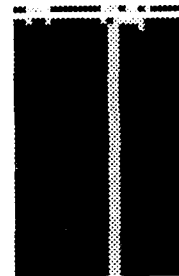
Rule 236



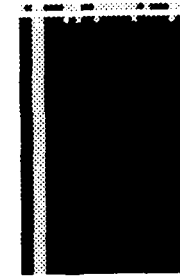
Rule 225



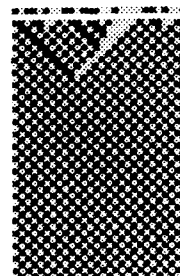
Rule 229



Rule 233



Rule 237



Rule 226



Rule 230



Rule 234



Rule 238



Rule 227



Rule 231

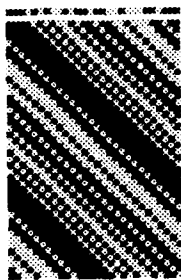


Rule 235

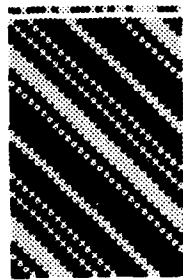


Rule 239

Figure B15. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.



Rule 240



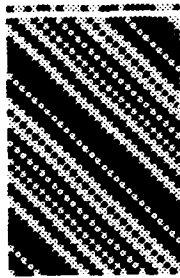
Rule 244



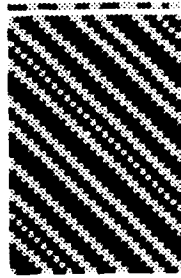
Rule 248



Rule 252



Rule 241



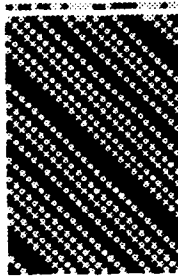
Rule 245



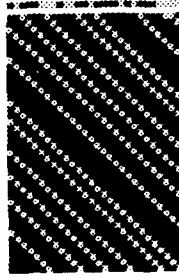
Rule 249



Rule 253



Rule 242



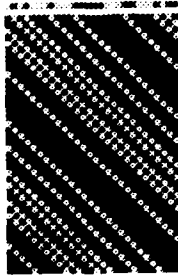
Rule 246



Rule 250



Rule 254



Rule 243



Rule 247



Rule 251



Rule 255

Figure B16. Space-time plots of nearest-neighbor cellular automata. Rule is defined in Appendix A.

APPENDIX C

ATTRACTOR SPECTRA FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

APPENDIX C

ATTRACTOR SPECTRA FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

This section catalogs the distribution of state attractors for nearest-neighbor coupled cellular automata. Limit cycle lengths are plotted versus the size of the simulated array. The ends of the linear array of cells were coupled together so that the cells formed a closed loop. That is, the edge cells are actually nearest neighbors in the simulations. Of the 256 possible nearest-neighbor rules, only 88 rule operations are independent. Since for example, rules 14,84,143, and 213 lead to identical attractor spectra, only Rule 14 is included in the plots. Refer to the Appendix A for a list of the unique rules and their mathematical representation.

The limiting dynamics falls into four simple classes of behavior, distinguishable by the various attractor types observed.

Class J Rules (Type A1 attractors) : 4, 5, 8, 12, 13, 19, 23, 28, 29, 32, 33, 36, 44, 50, 51, 72, 76, 77, 78, 104, 128

Class K Rules (Types A1 and A2 attractors) : 3, 6, 7, 9, 10, 11, 14, 15, 24, 27, 40, 42, 43, 46, 56, 57, 58, 62, 74, 130, 134, 138, 142, 152, 162, 184

Class L Rules (Types A1 and A3 attractors) : 8, 30, 37, 45, 54, 60, 73, 90, 94, 105, 110, 122, 126, 146, 150, 164, 168, 170, 172

Class M Rules (Types A1, A2, and A3 attractors) : 25, 26, 35, 38, 41, 106

where we have defined the attractor types as follows.

Type A1 Attractor : Cycle length is independent of array size.

Type A2 Attractor : Cycle length is in integral proportion to array size.

Type A3 Attractor : Cycle length is uncorrelated with array size.

RULE 2

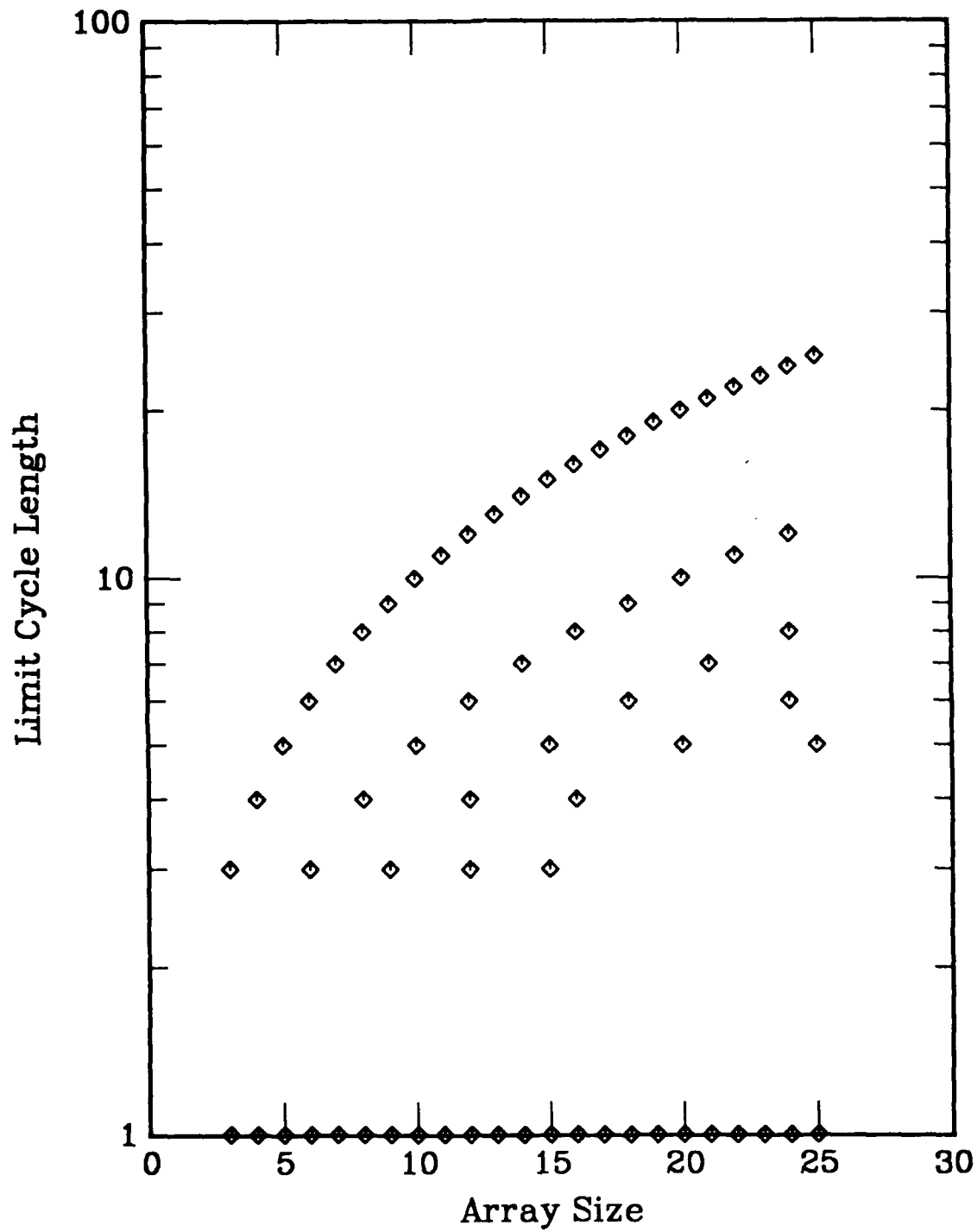


Figure C1. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 3

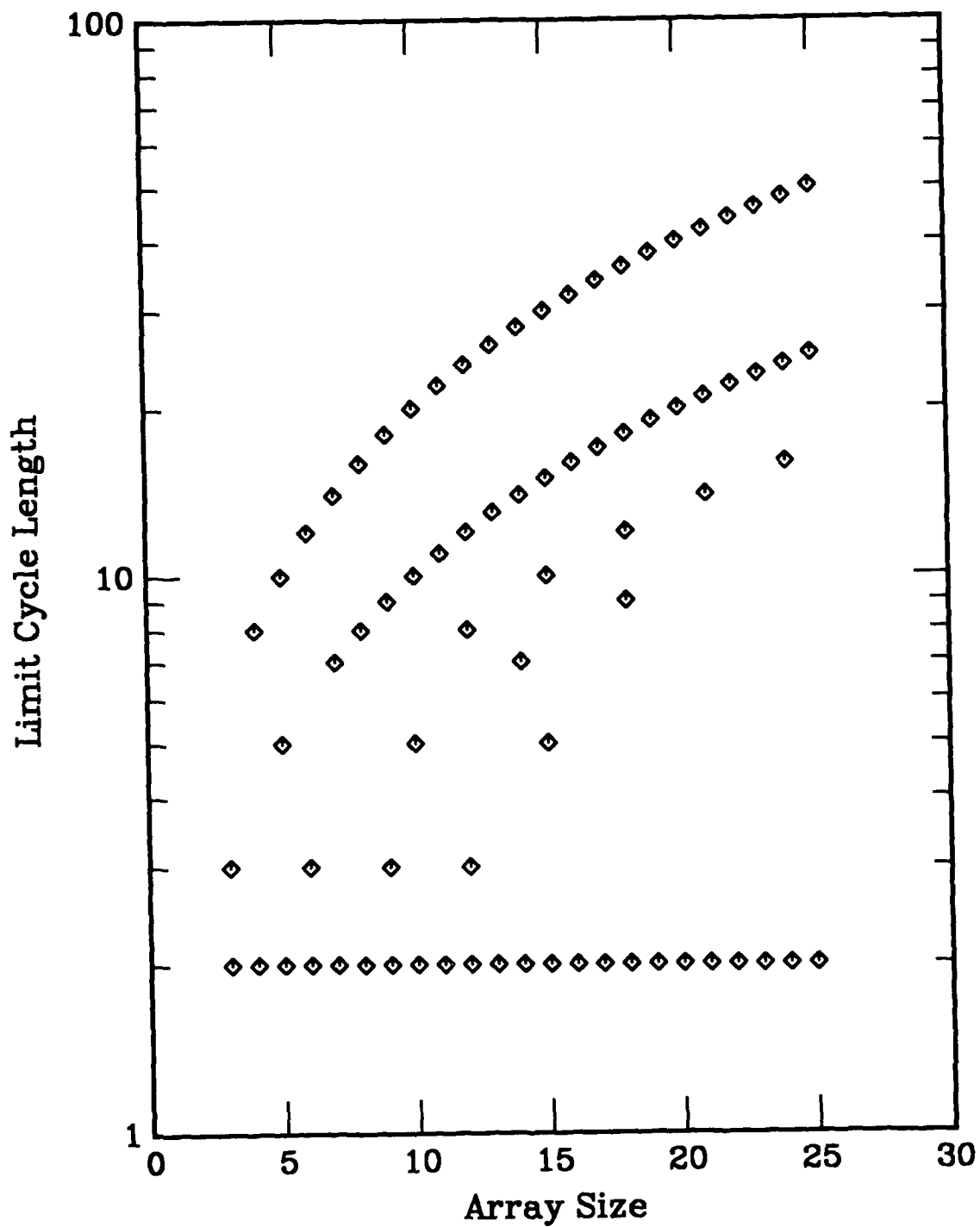


Figure C2. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

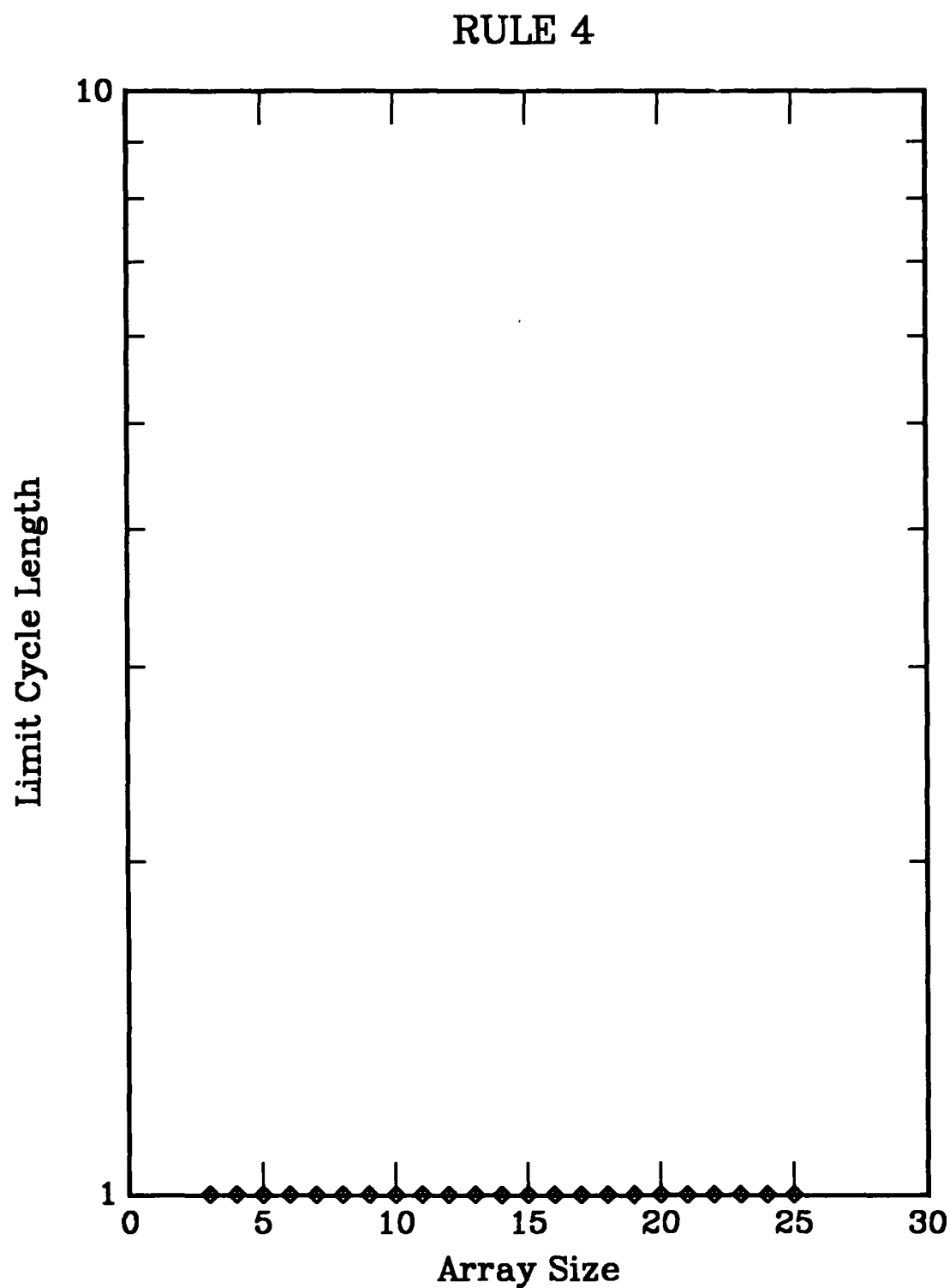


Figure C3. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 5

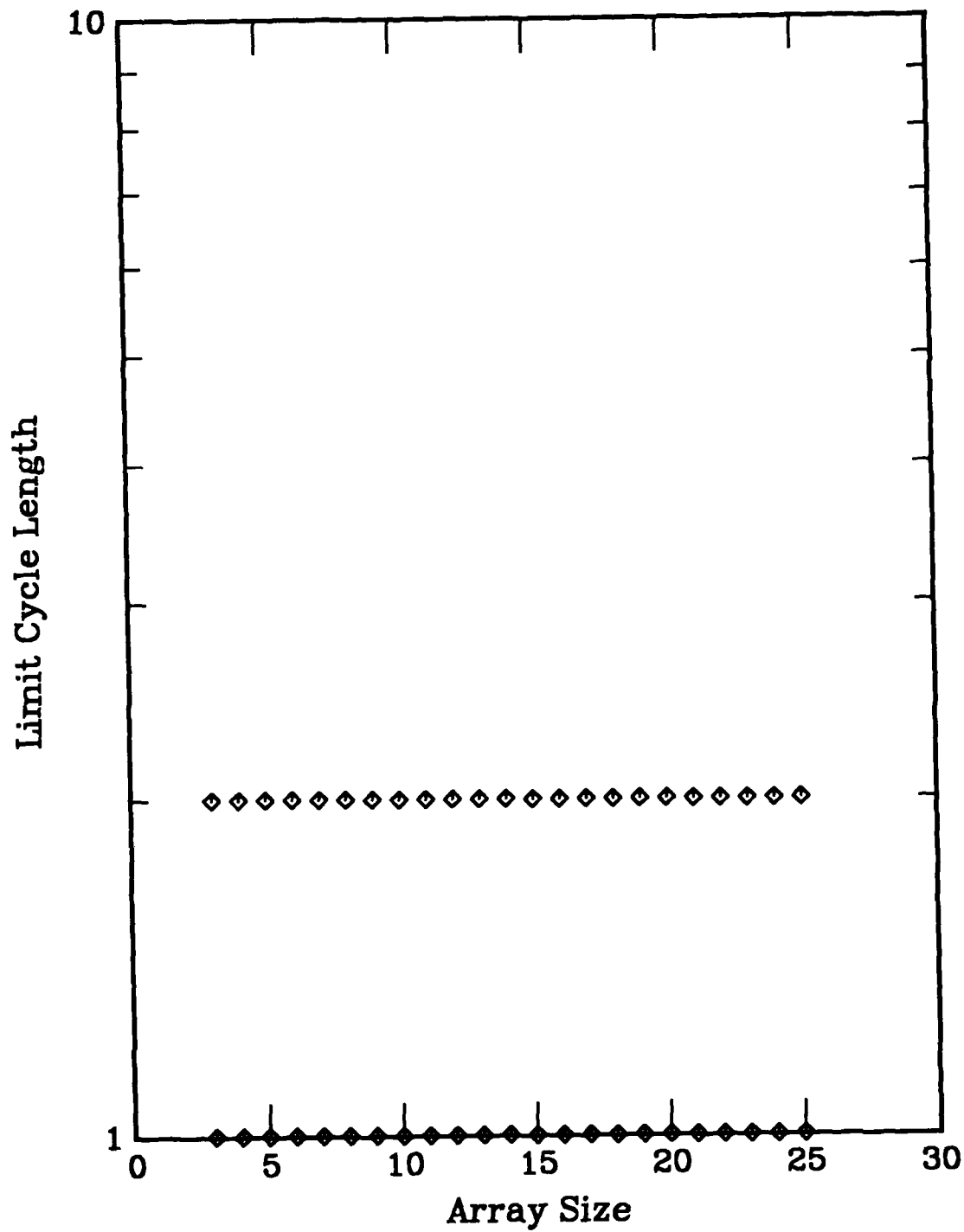


Figure C4. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 6

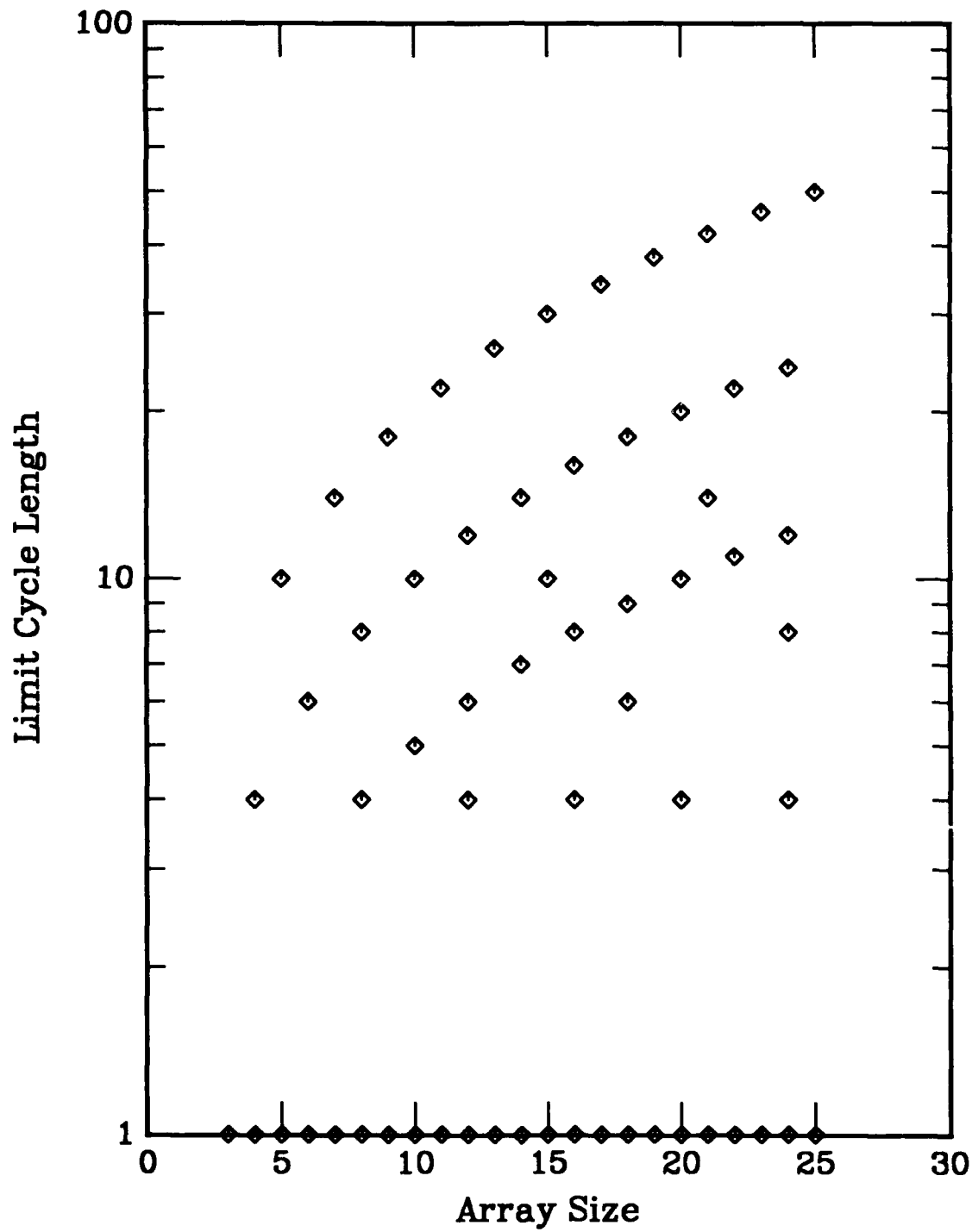


Figure C5. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 7

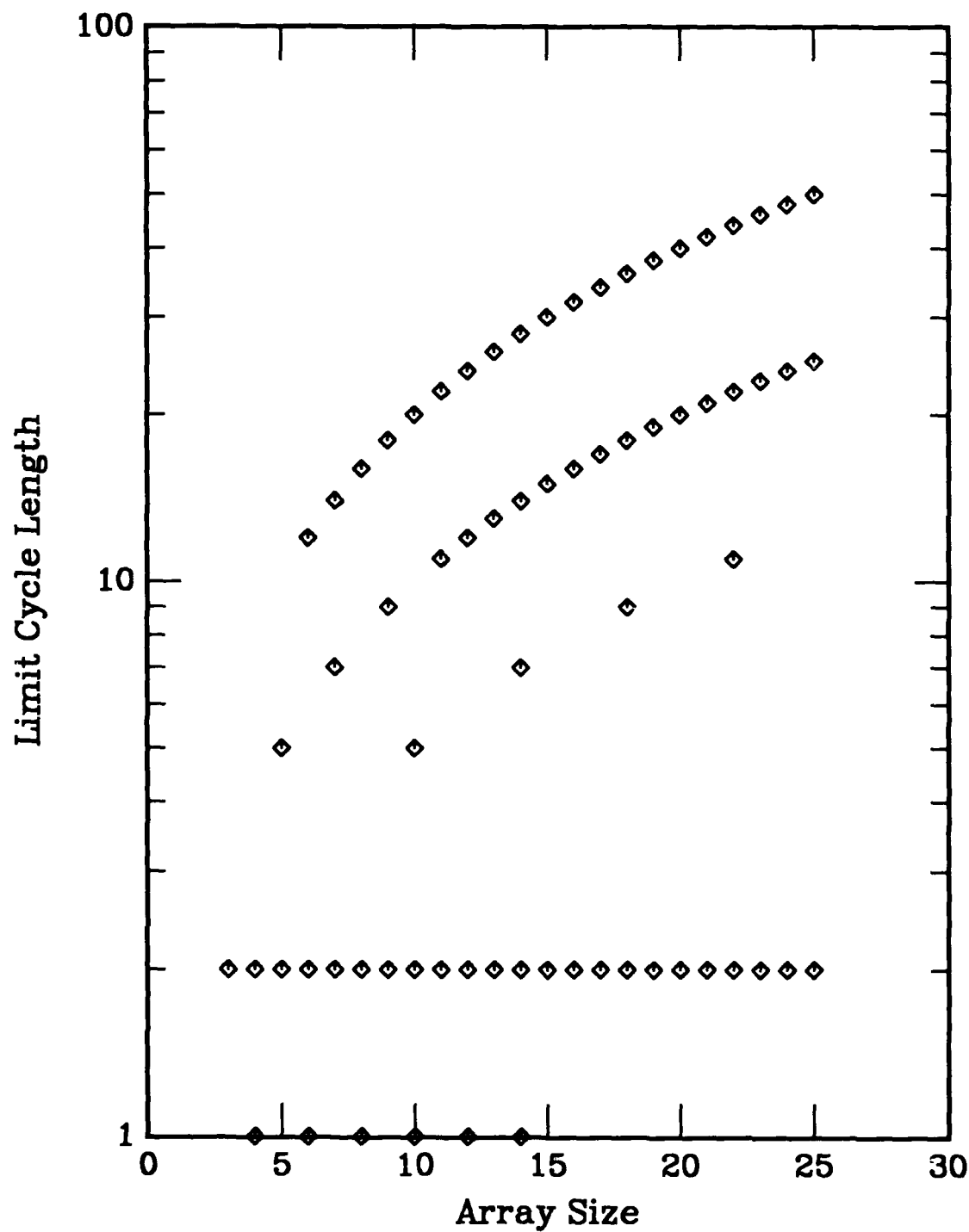


Figure C6. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

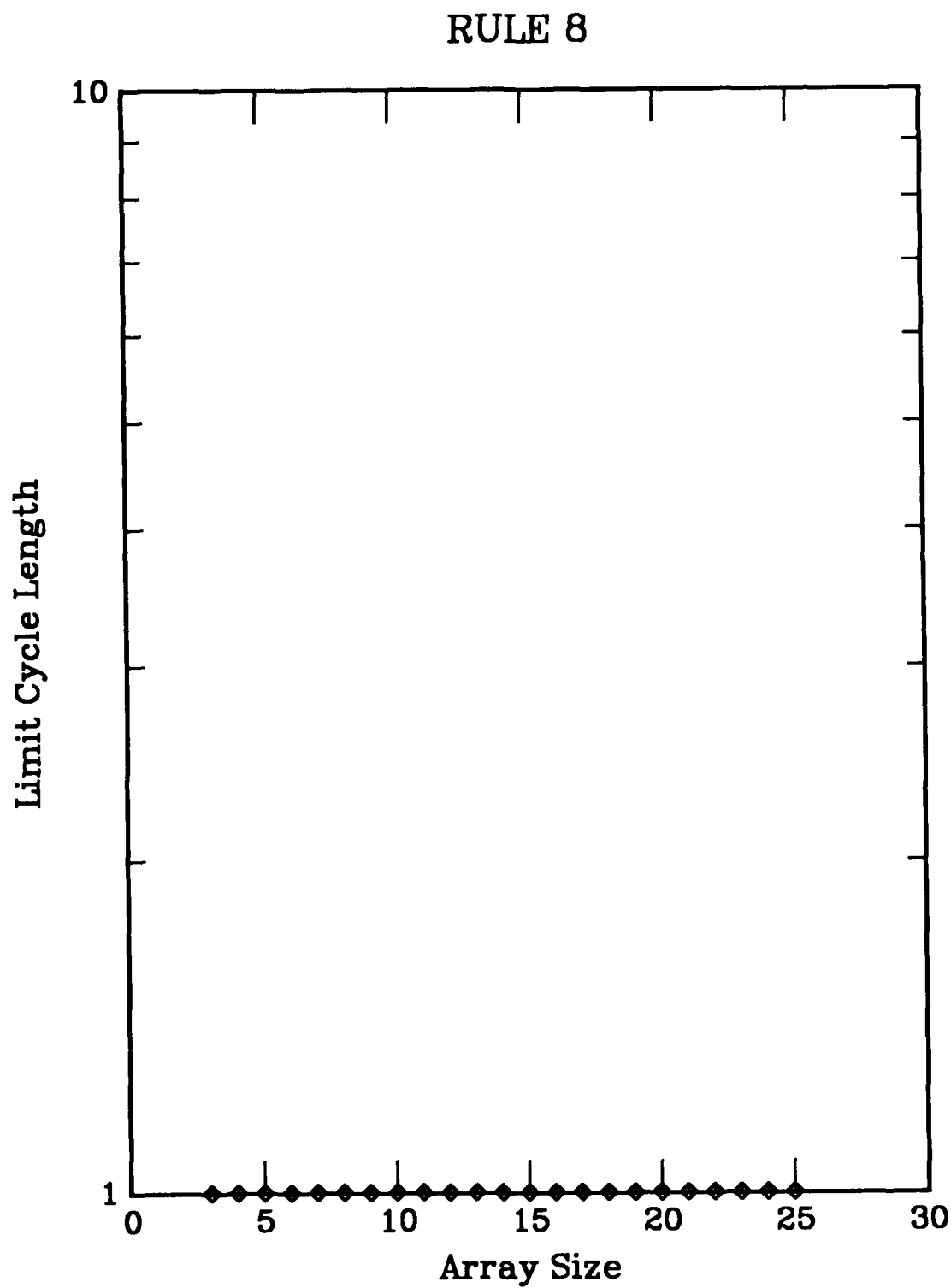


Figure C7. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 9

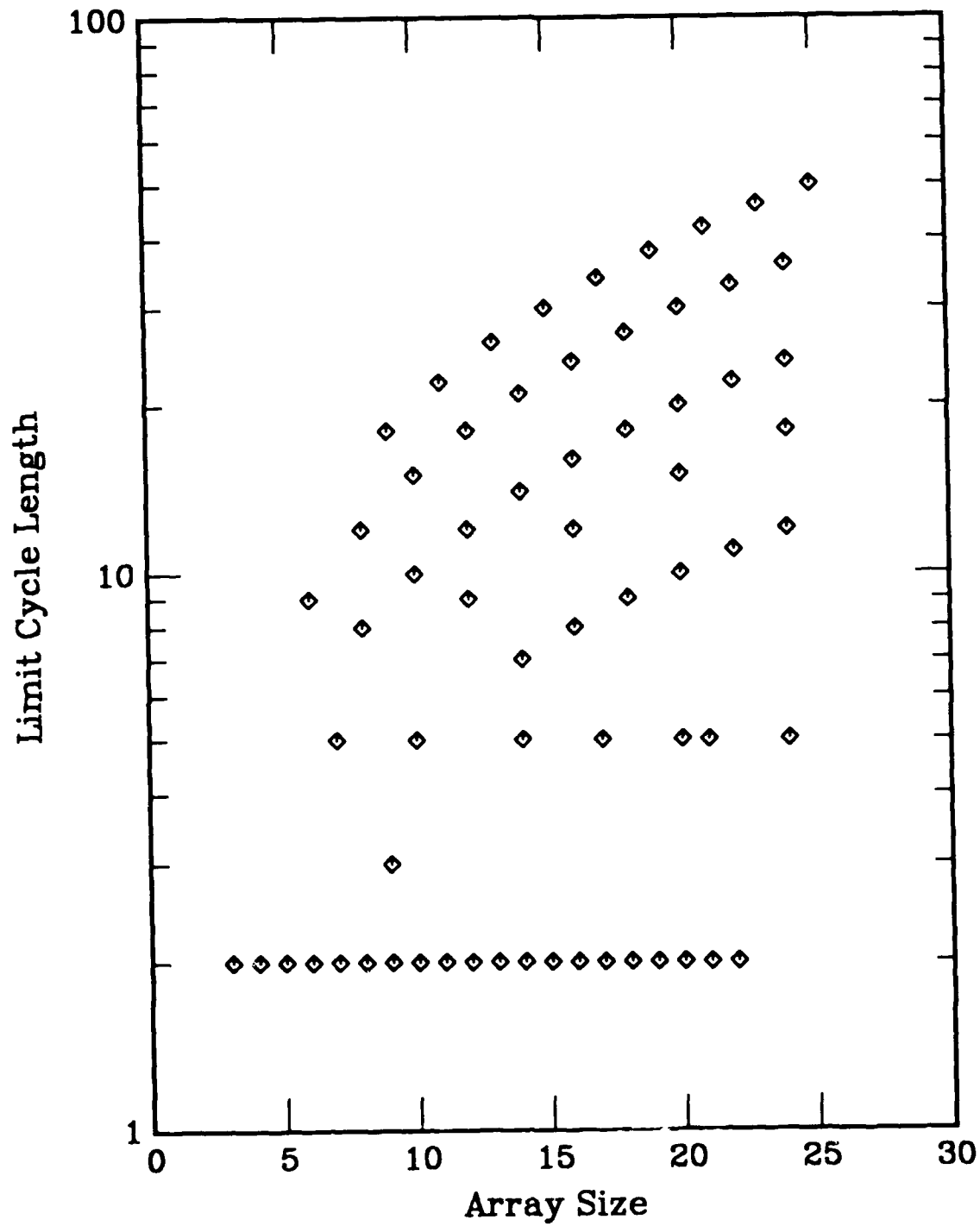


Figure C8. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

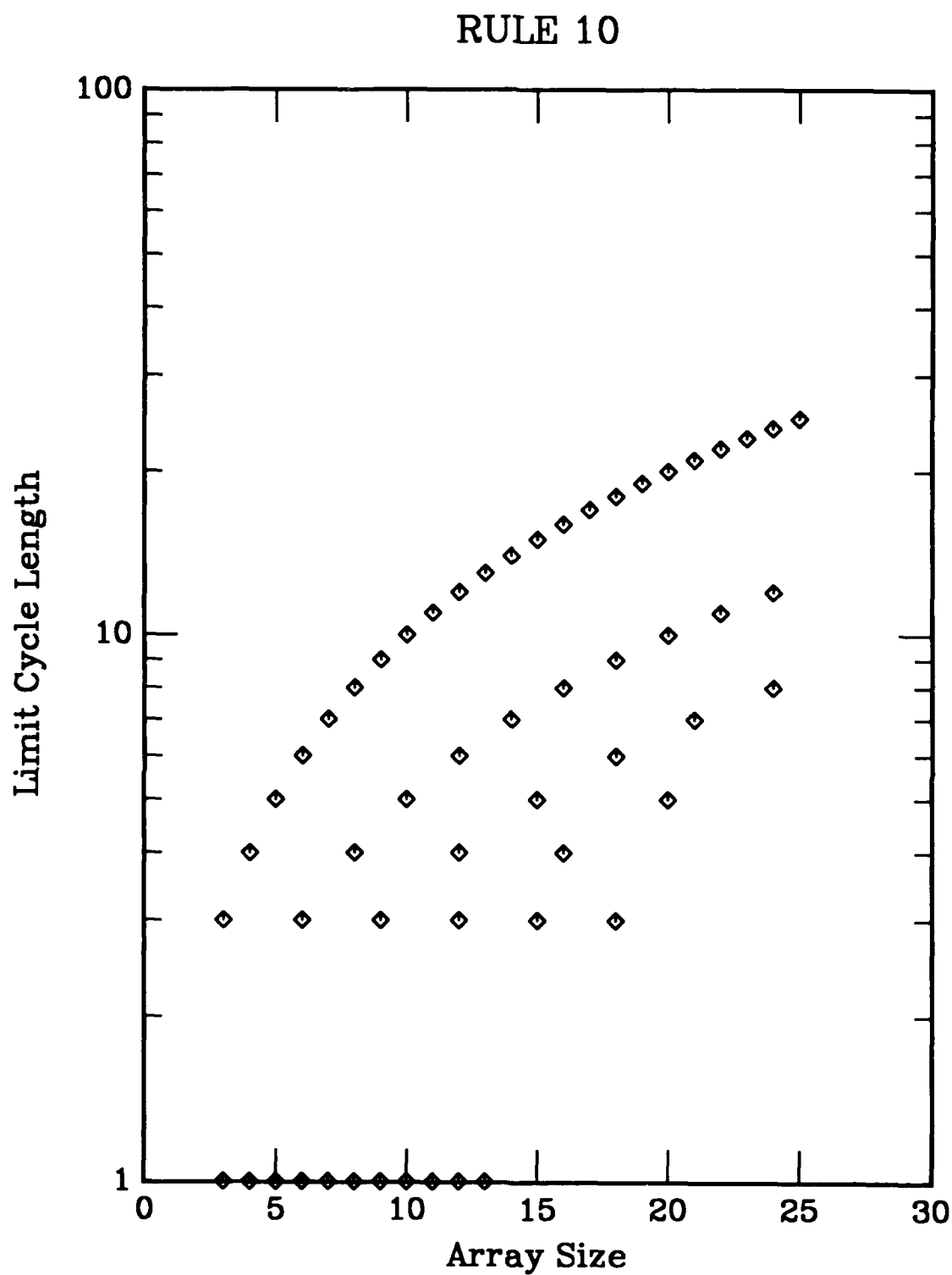


Figure C9. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 11

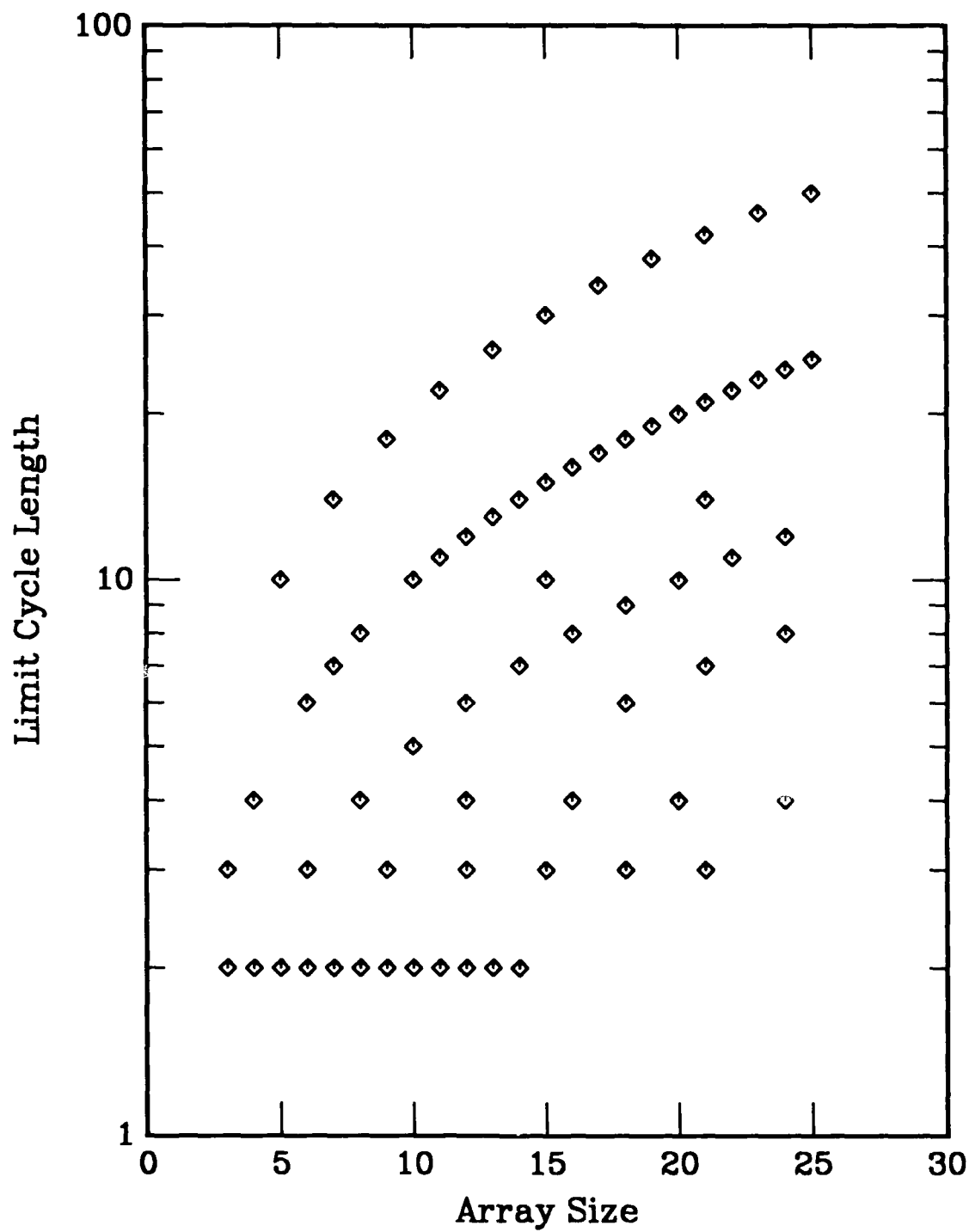


Figure C10. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

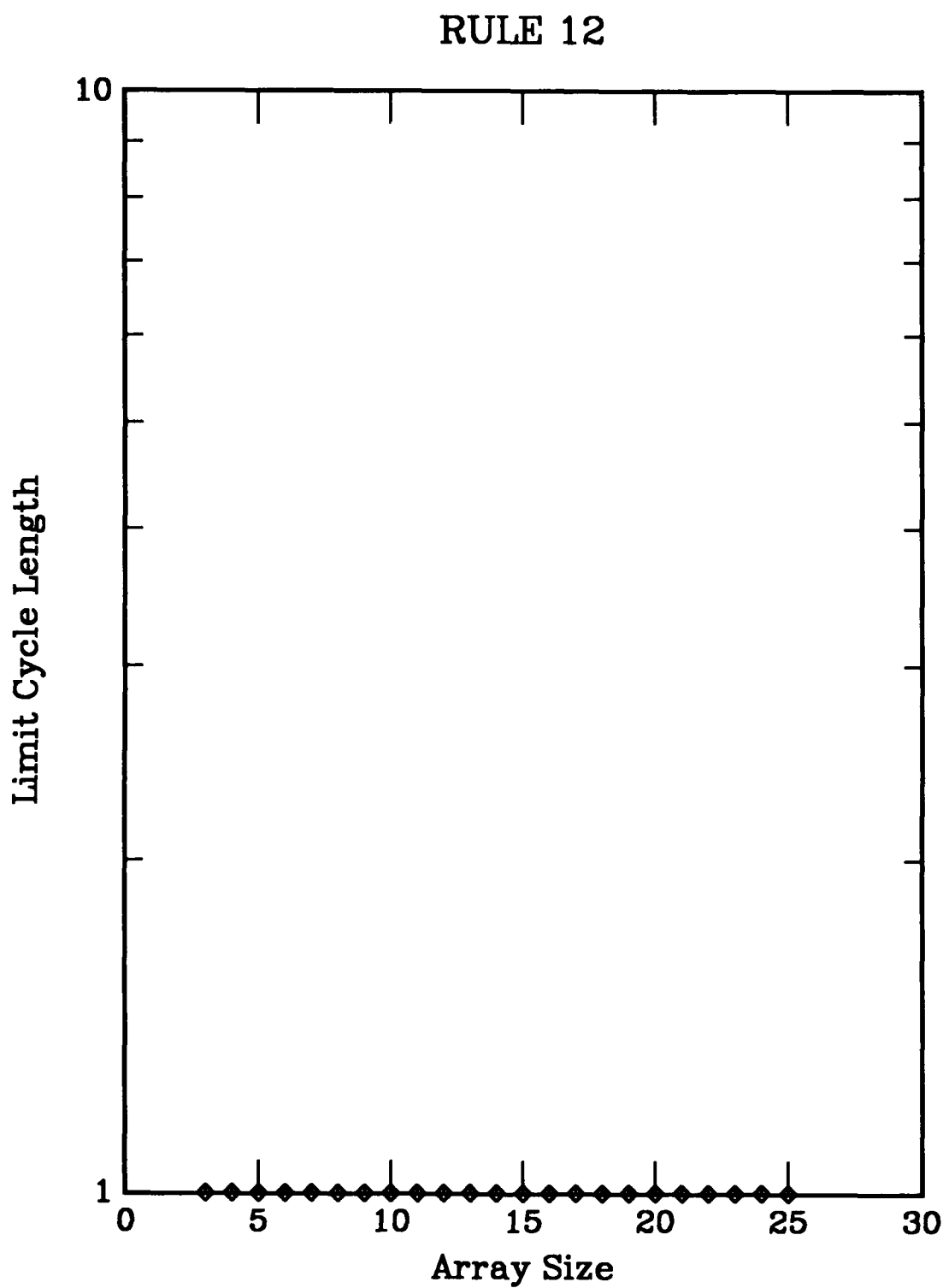


Figure C11. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 13

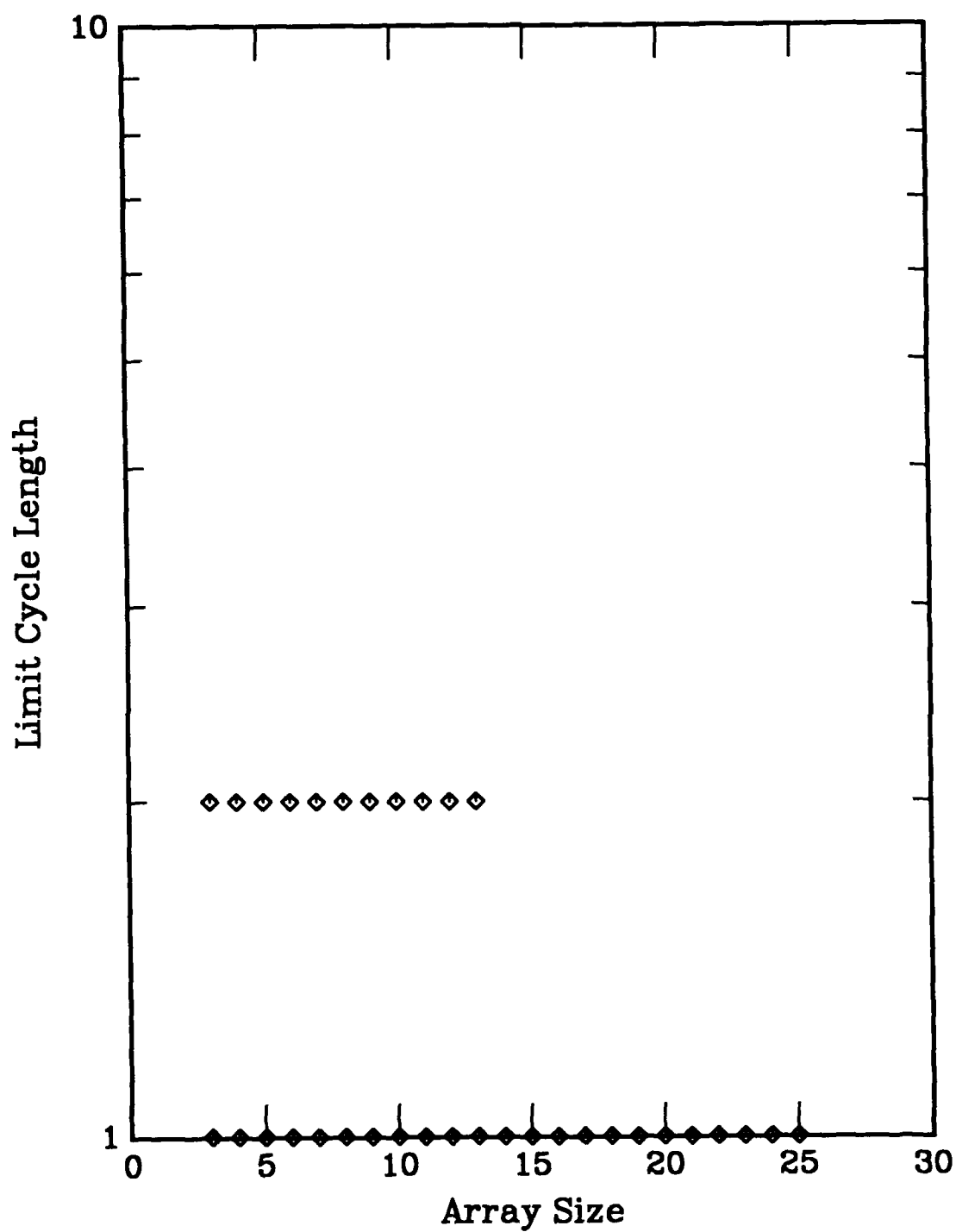


Figure C12. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 14

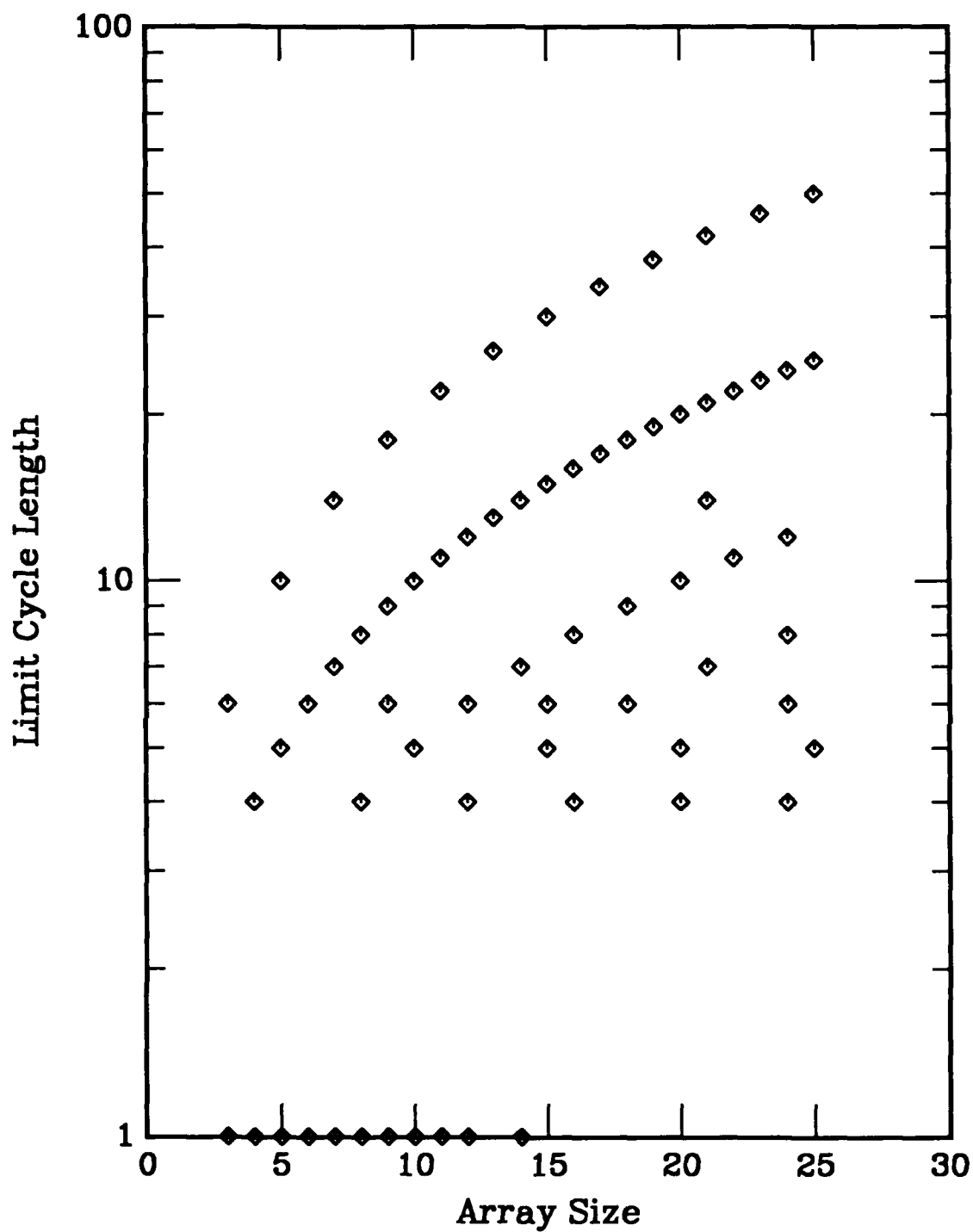


Figure C13. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 15

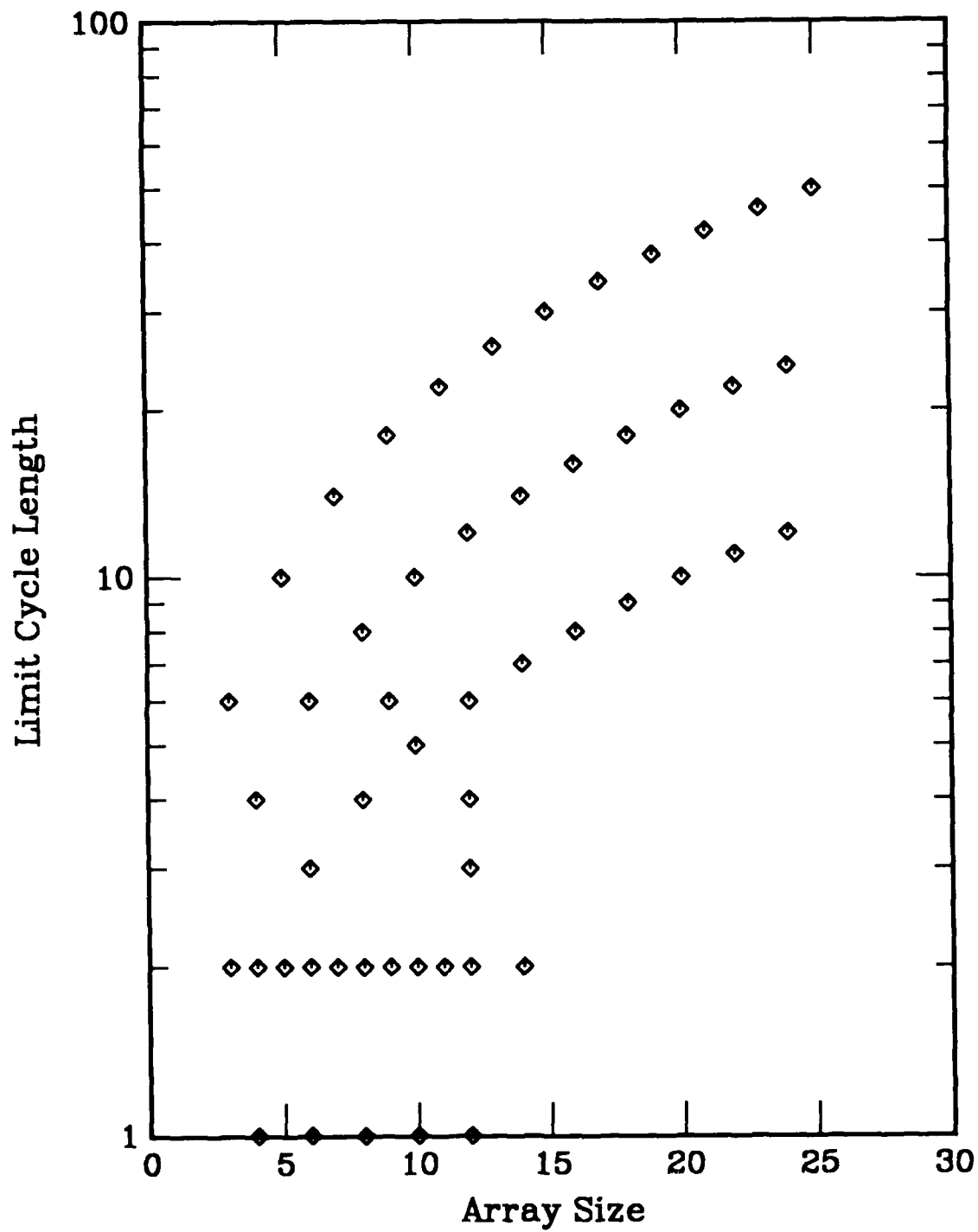


Figure C14. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 18

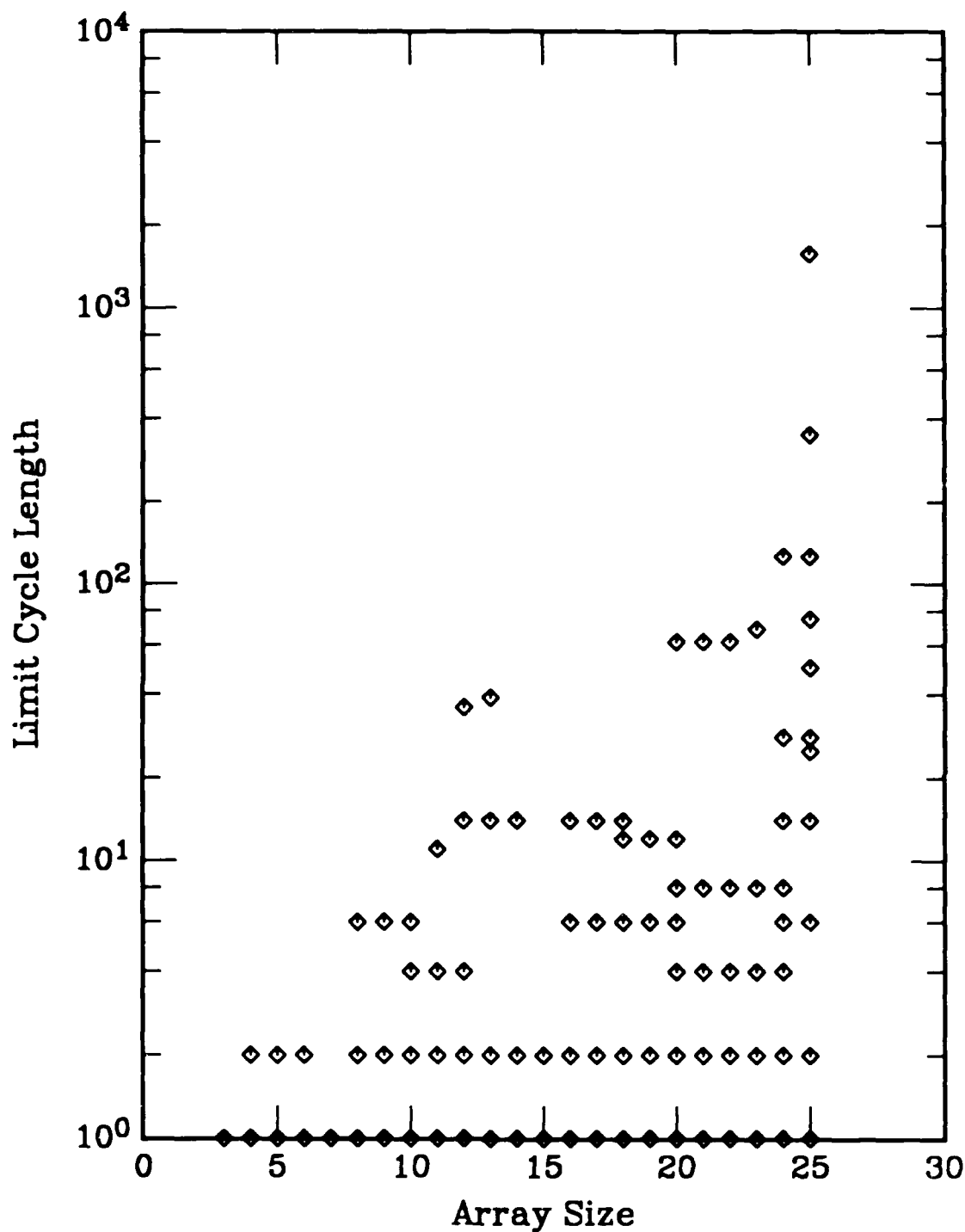


Figure C15. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 19

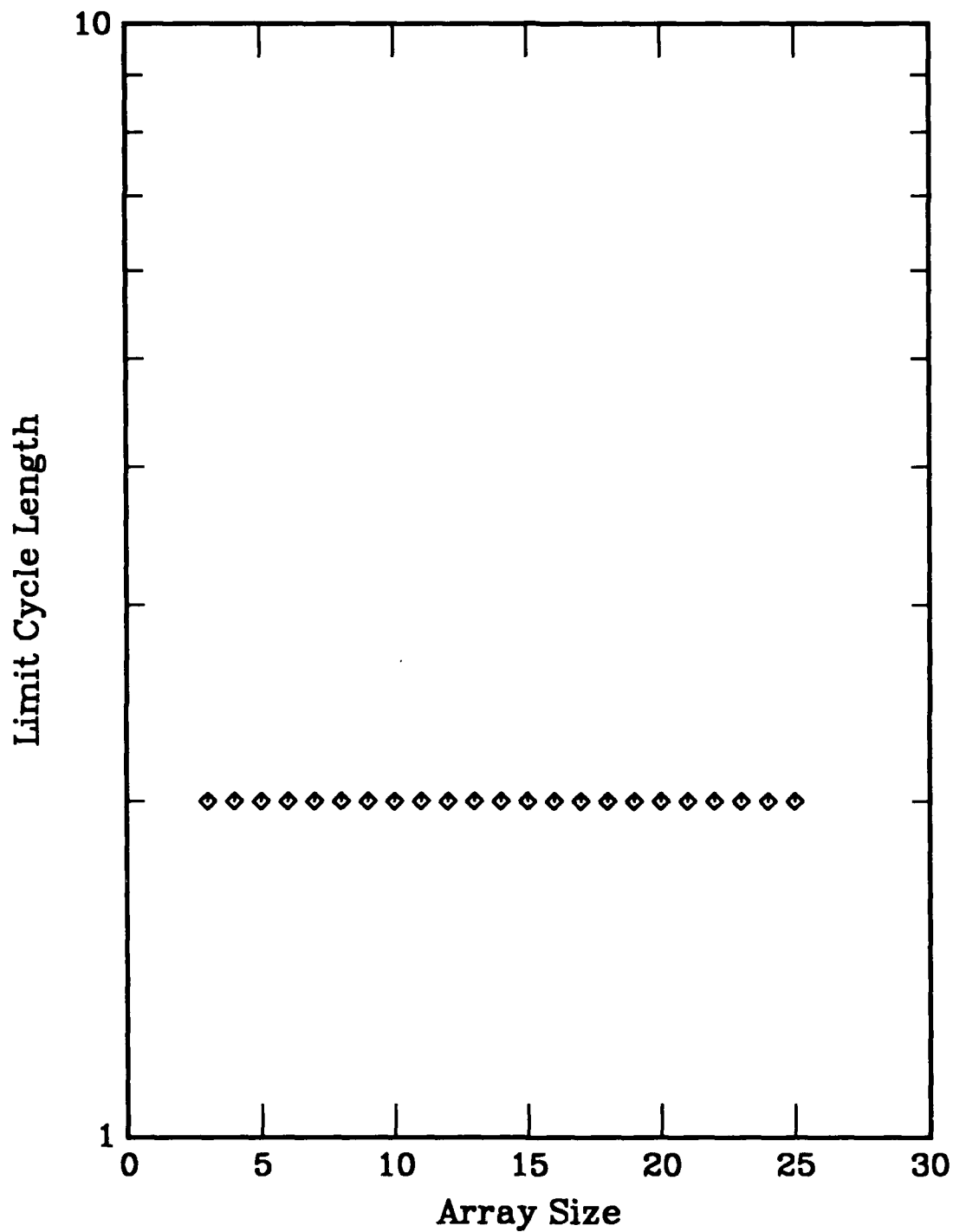


Figure C16. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 23

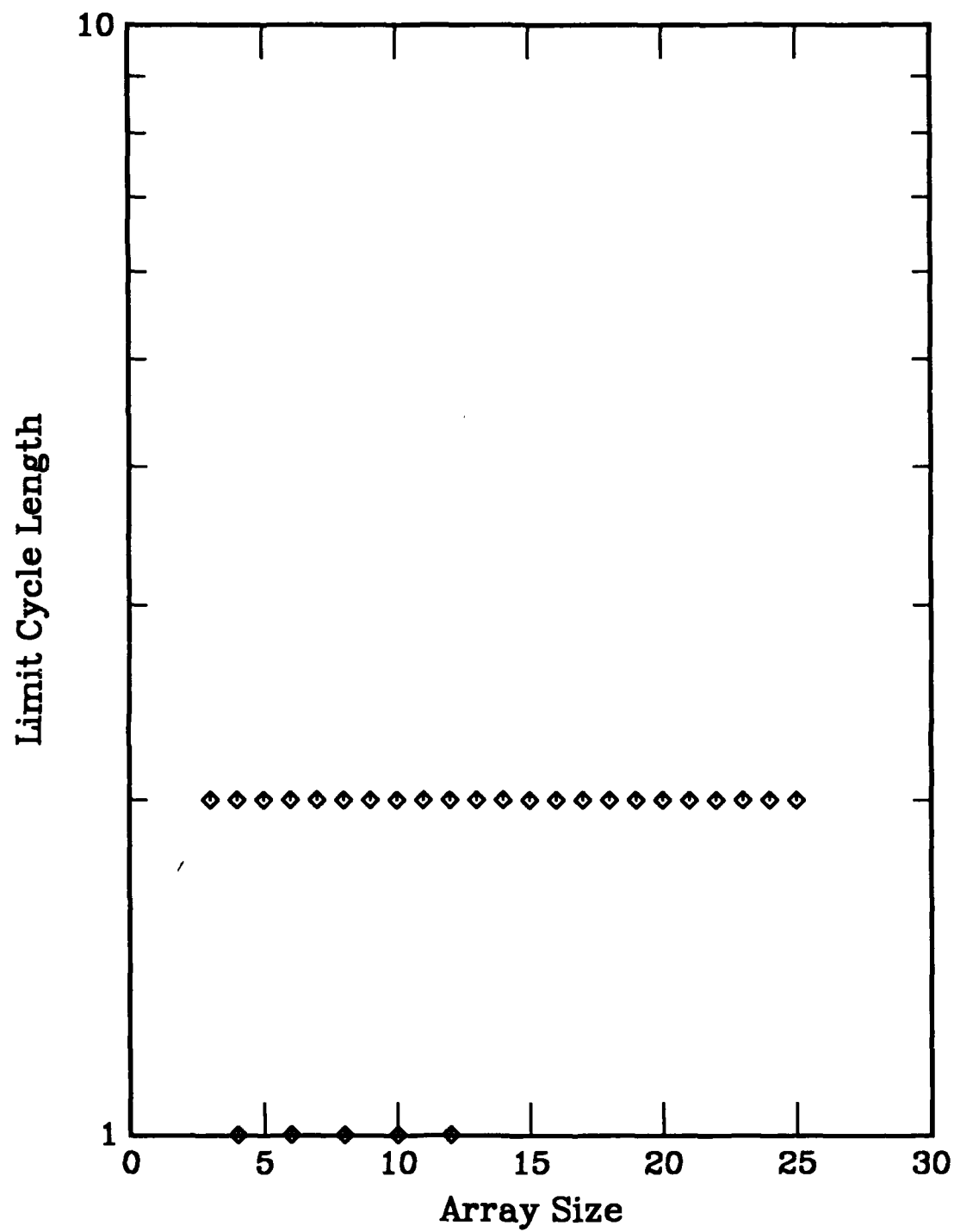


Figure C17. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 24

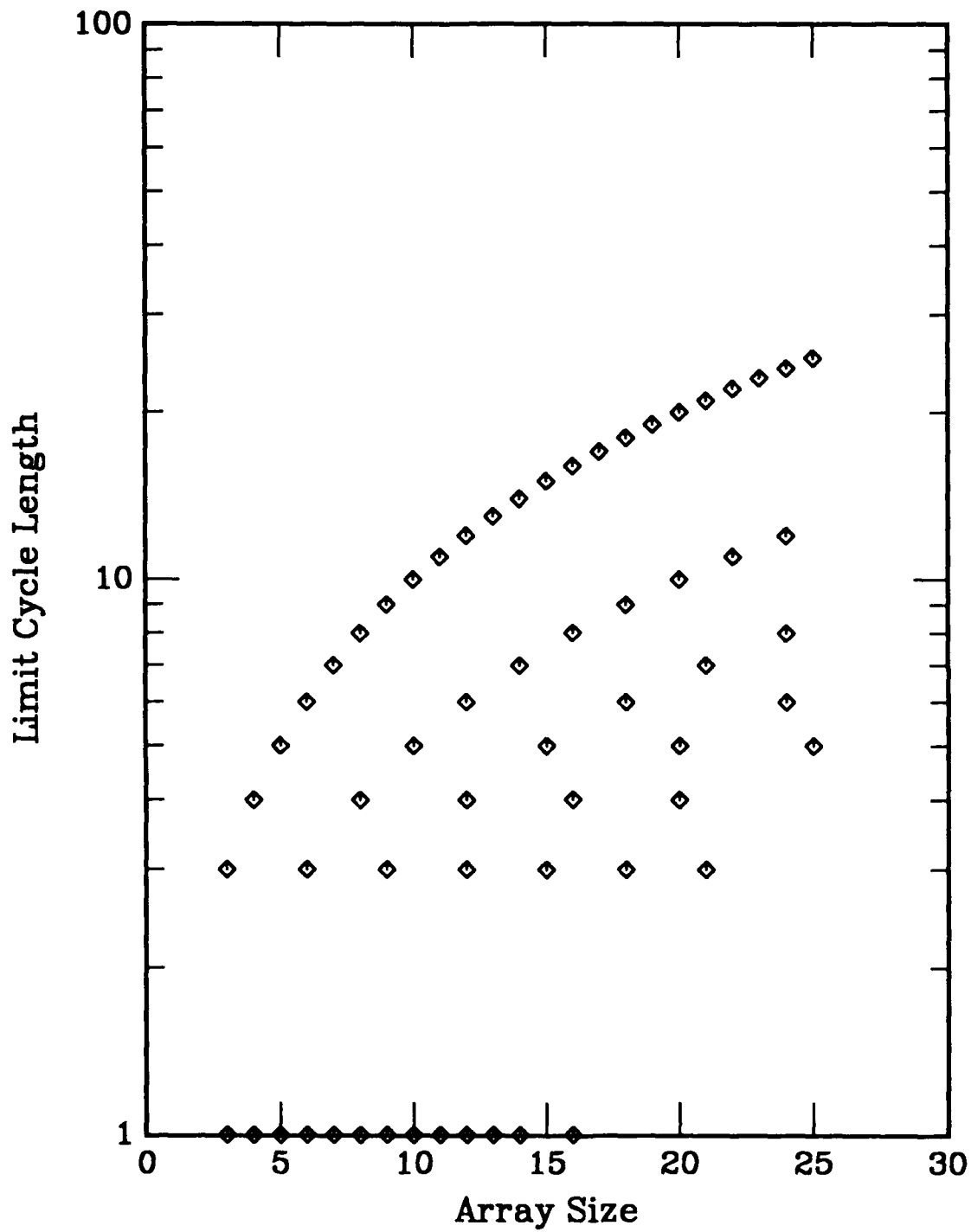


Figure C18. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 25

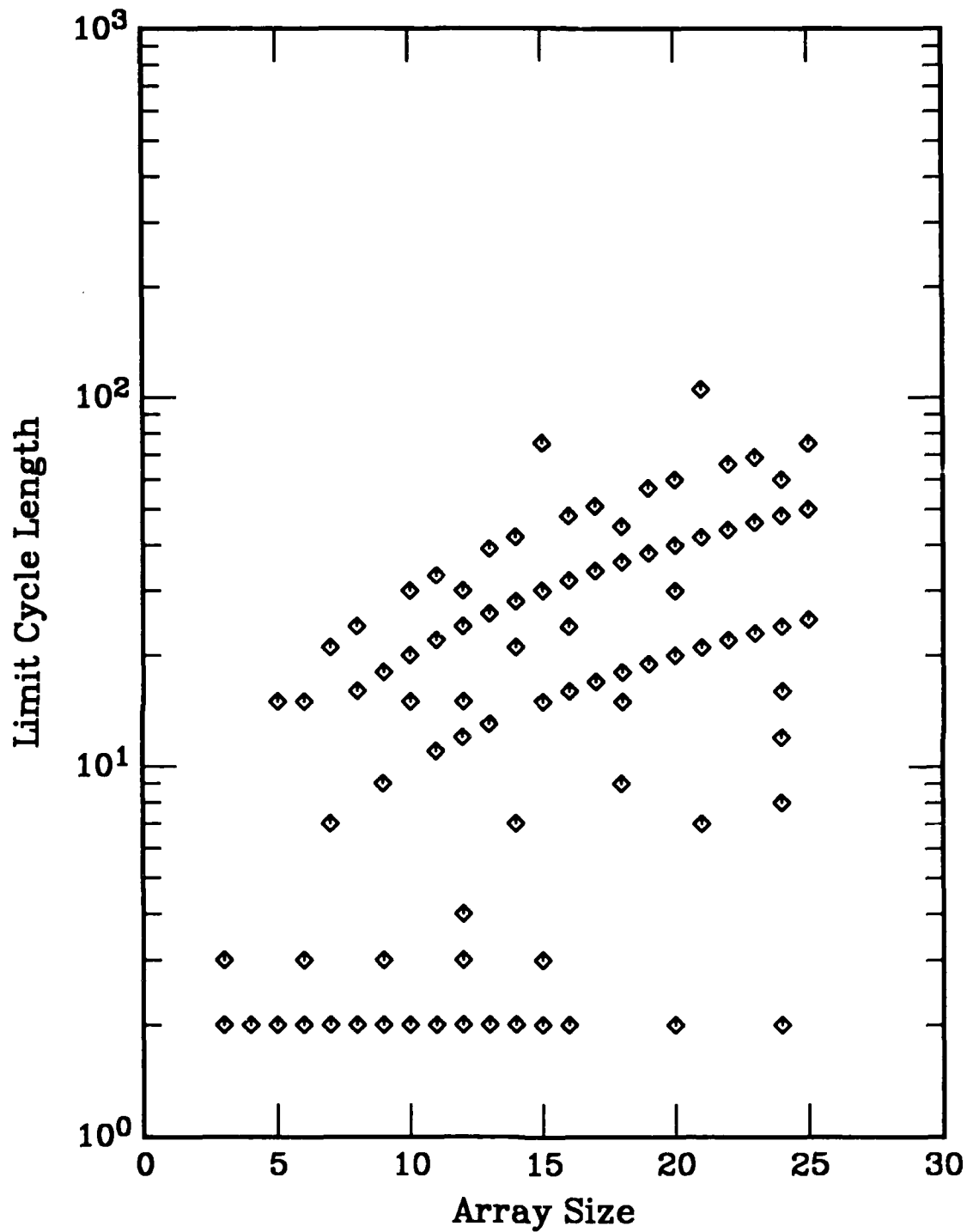


Figure C19. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 26

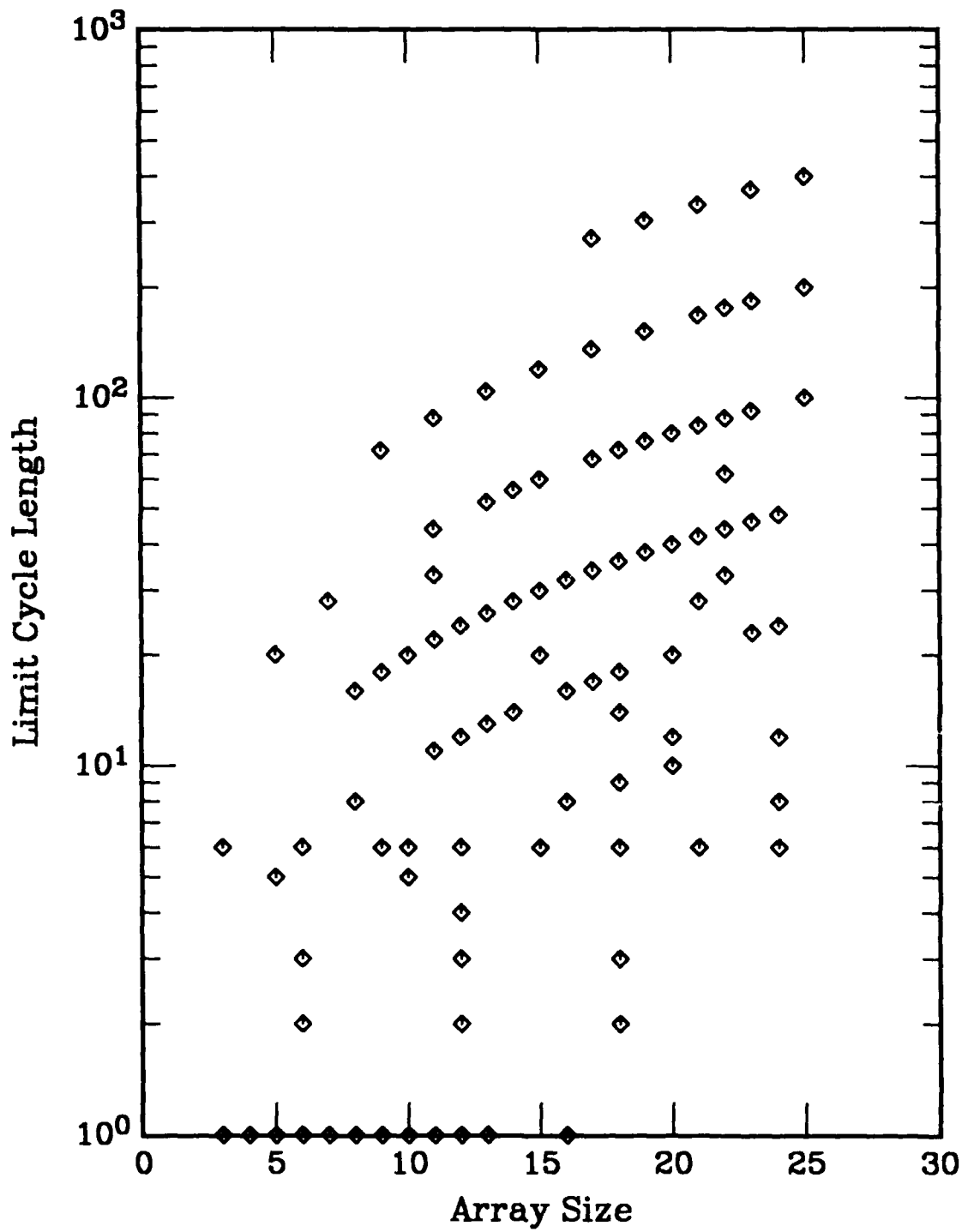


Figure C20. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

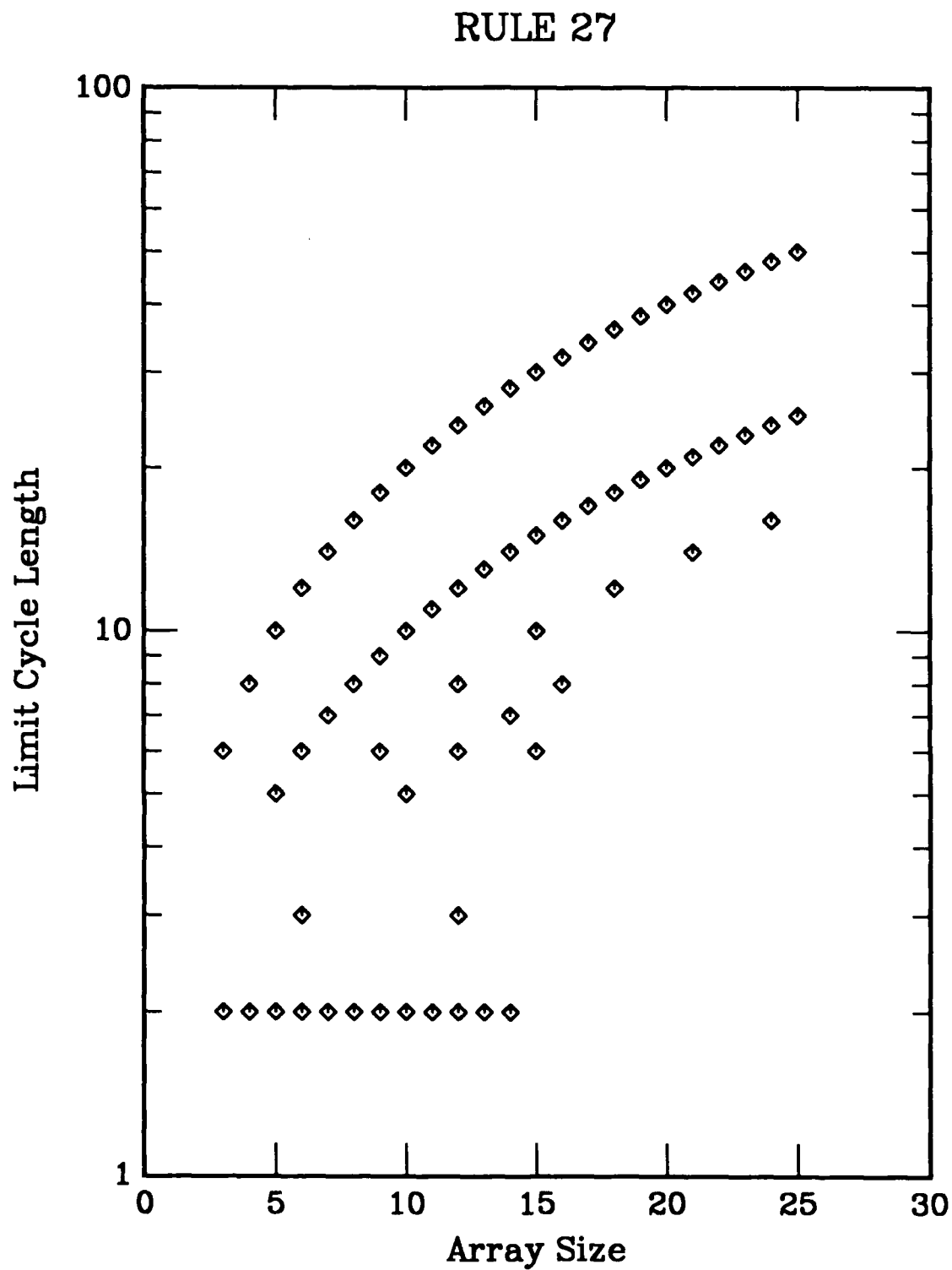


Figure C21. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 28

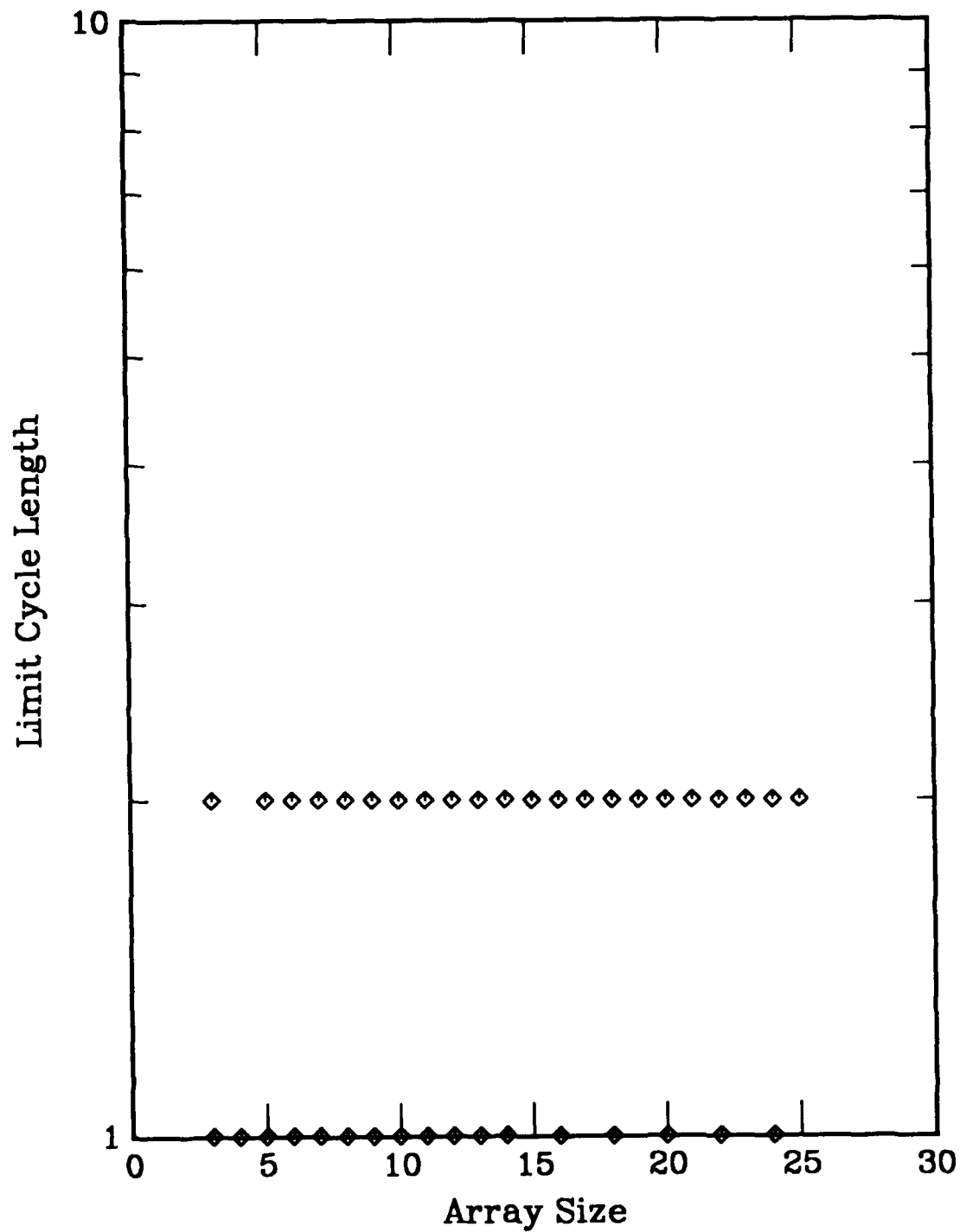


Figure C22. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 29

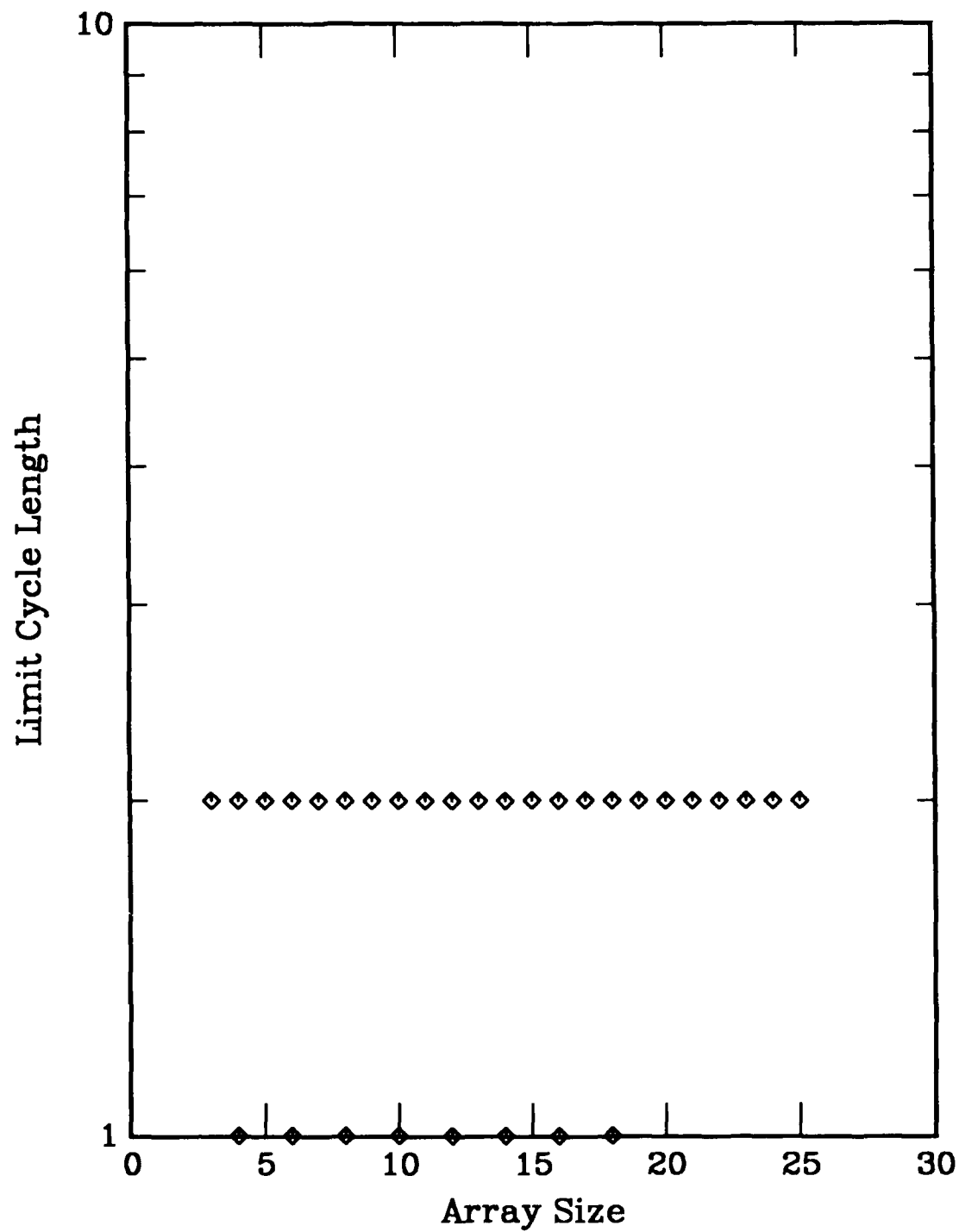


Figure C23. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 30

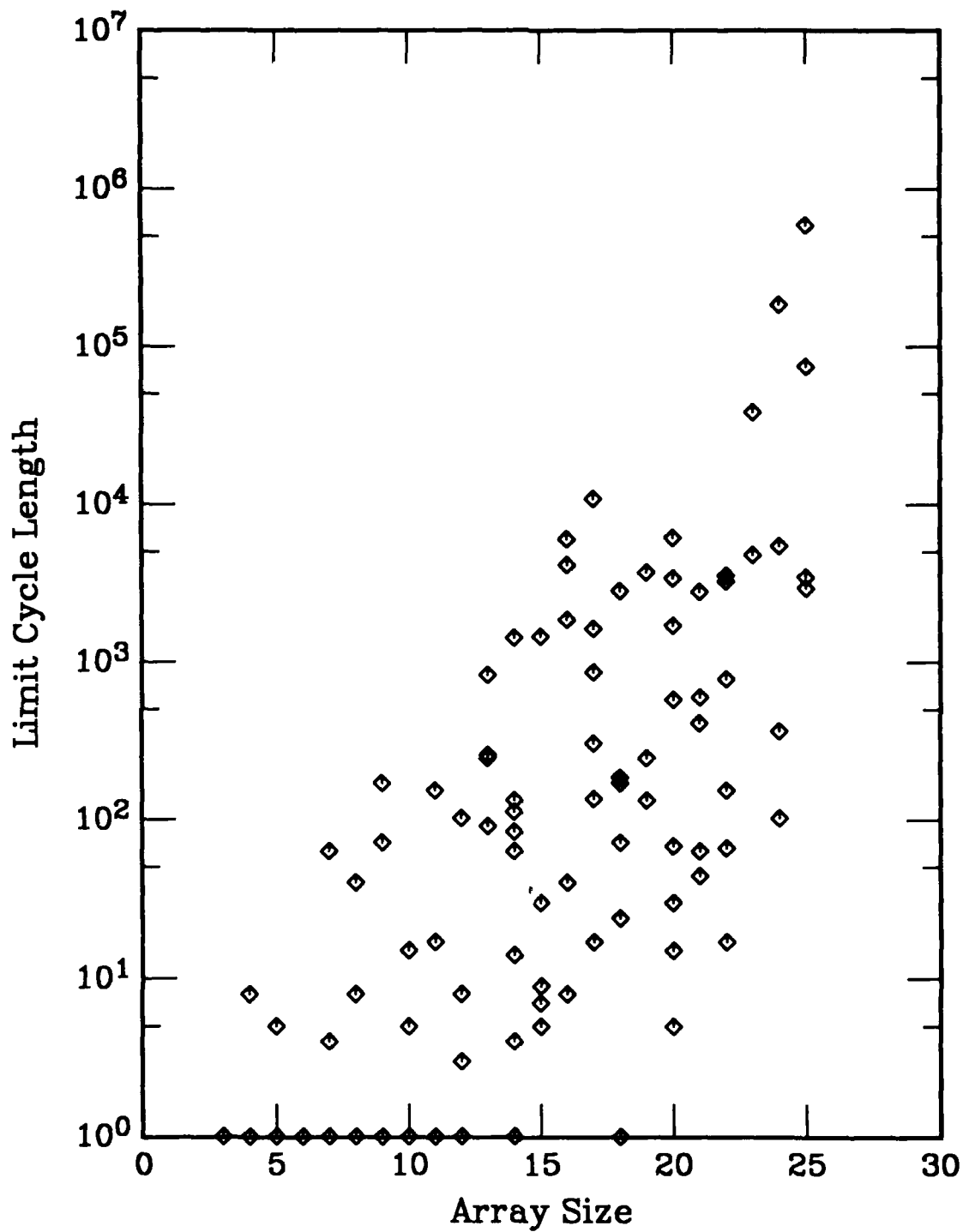


Figure C24. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

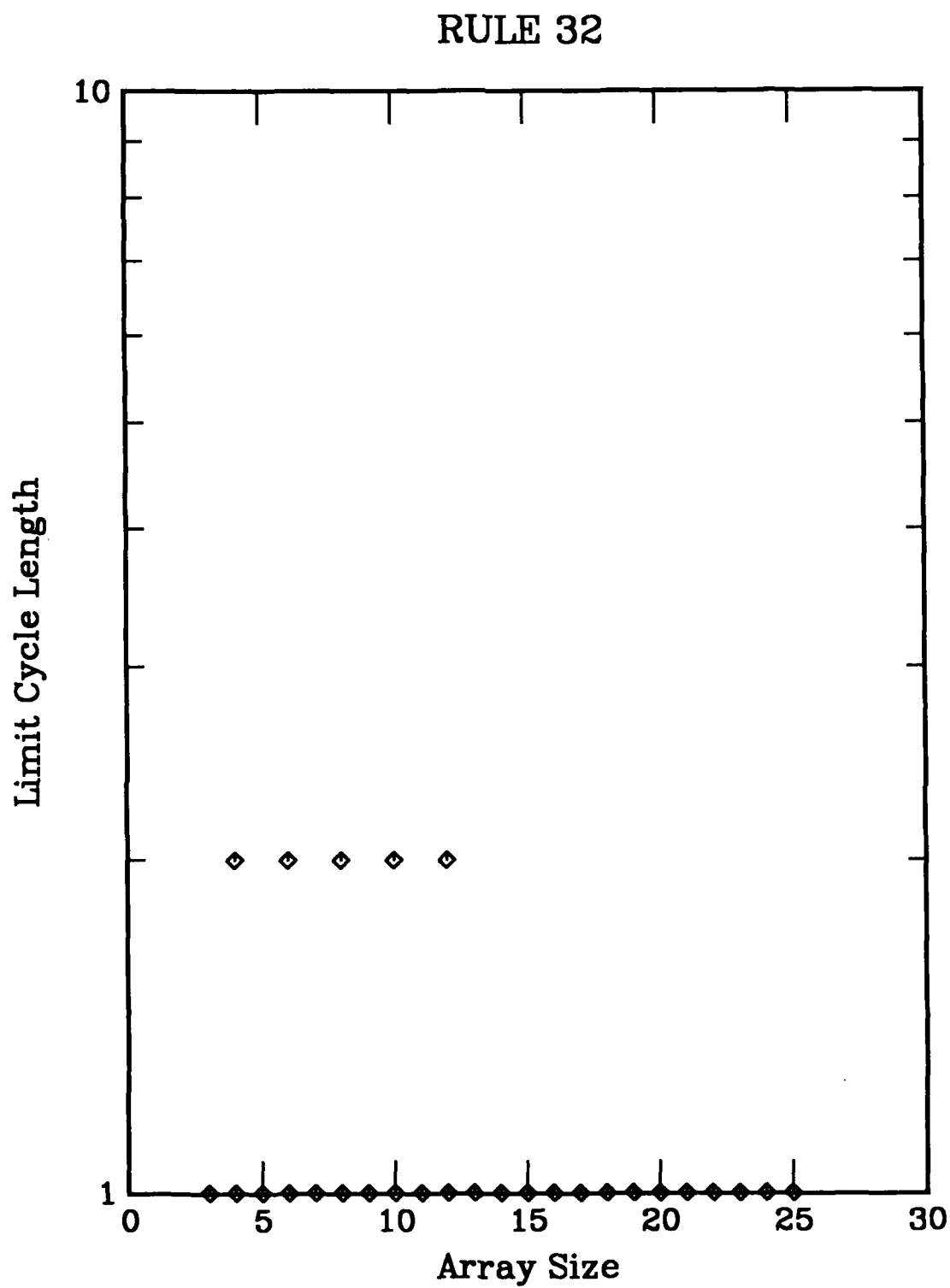


Figure C25. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 33

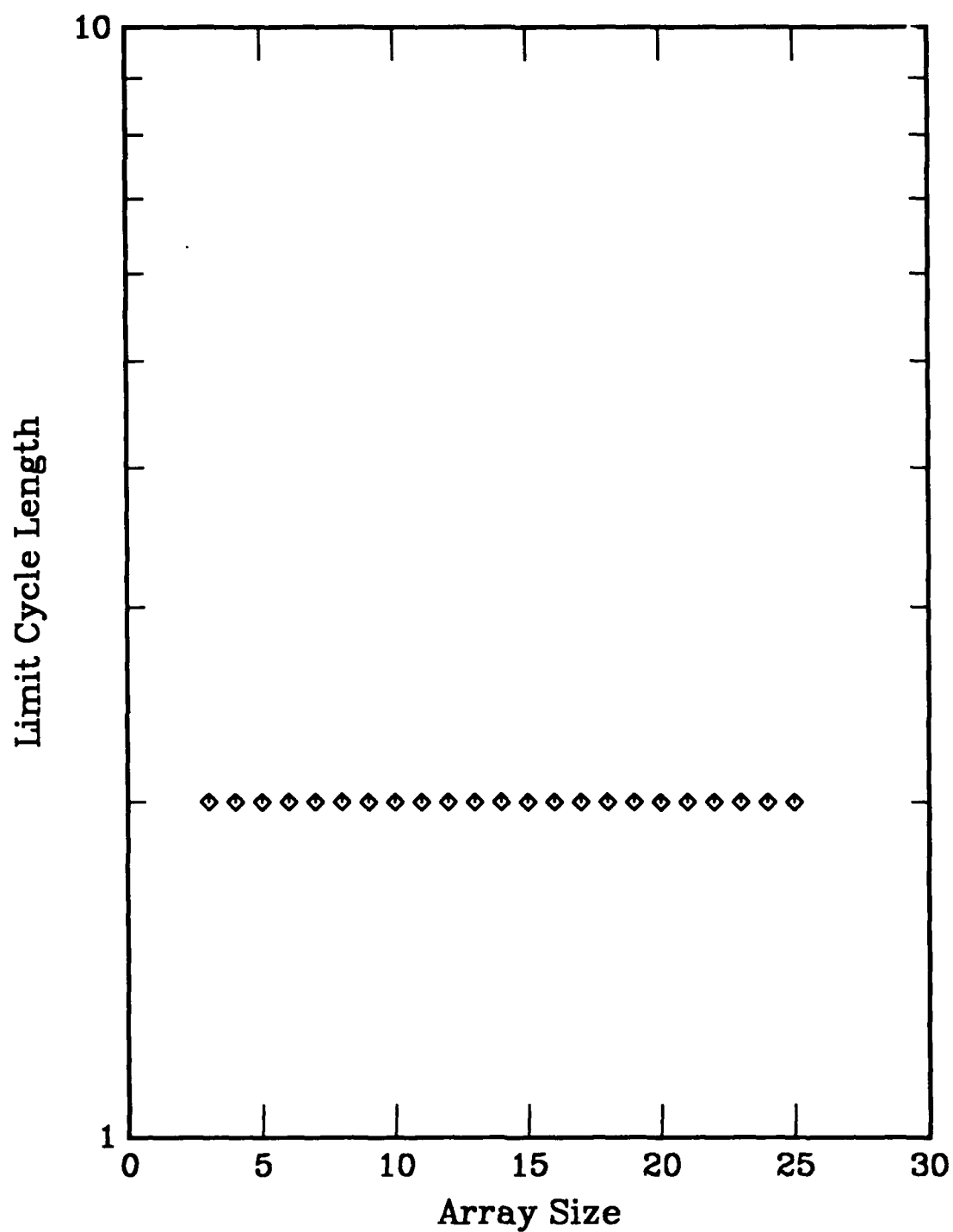


Figure C26. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 34

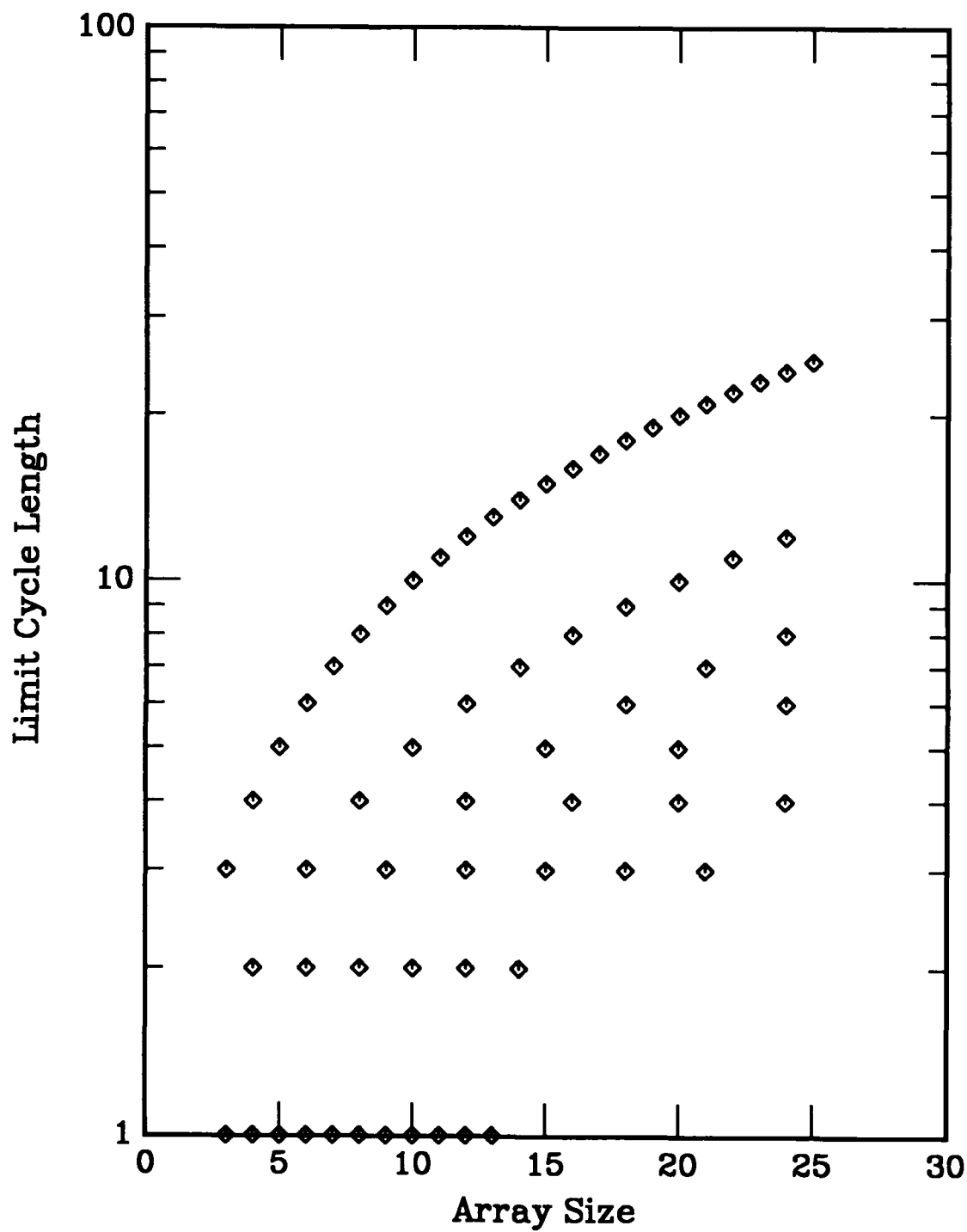


Figure C27. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 35

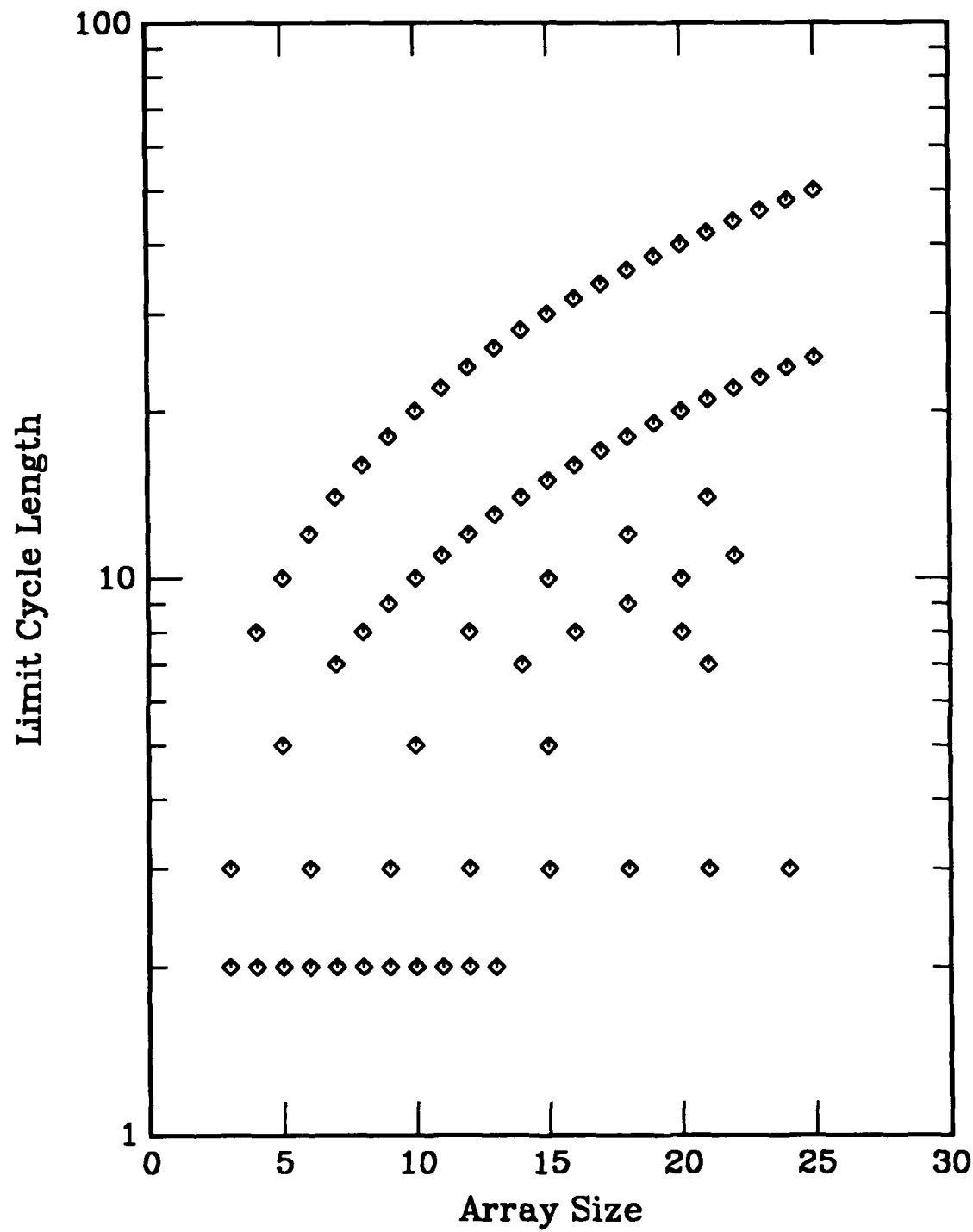


Figure C28. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 36

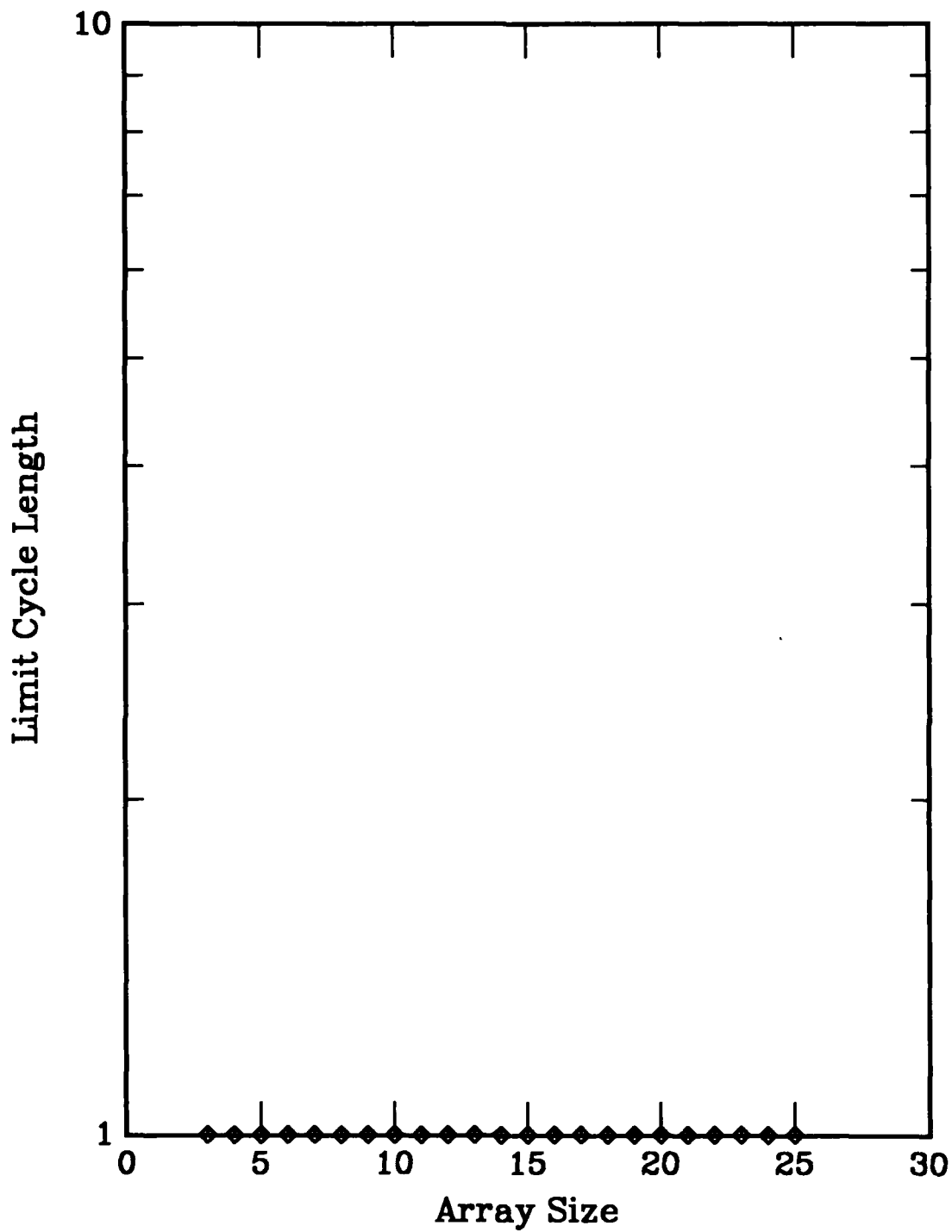


Figure C29. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 37

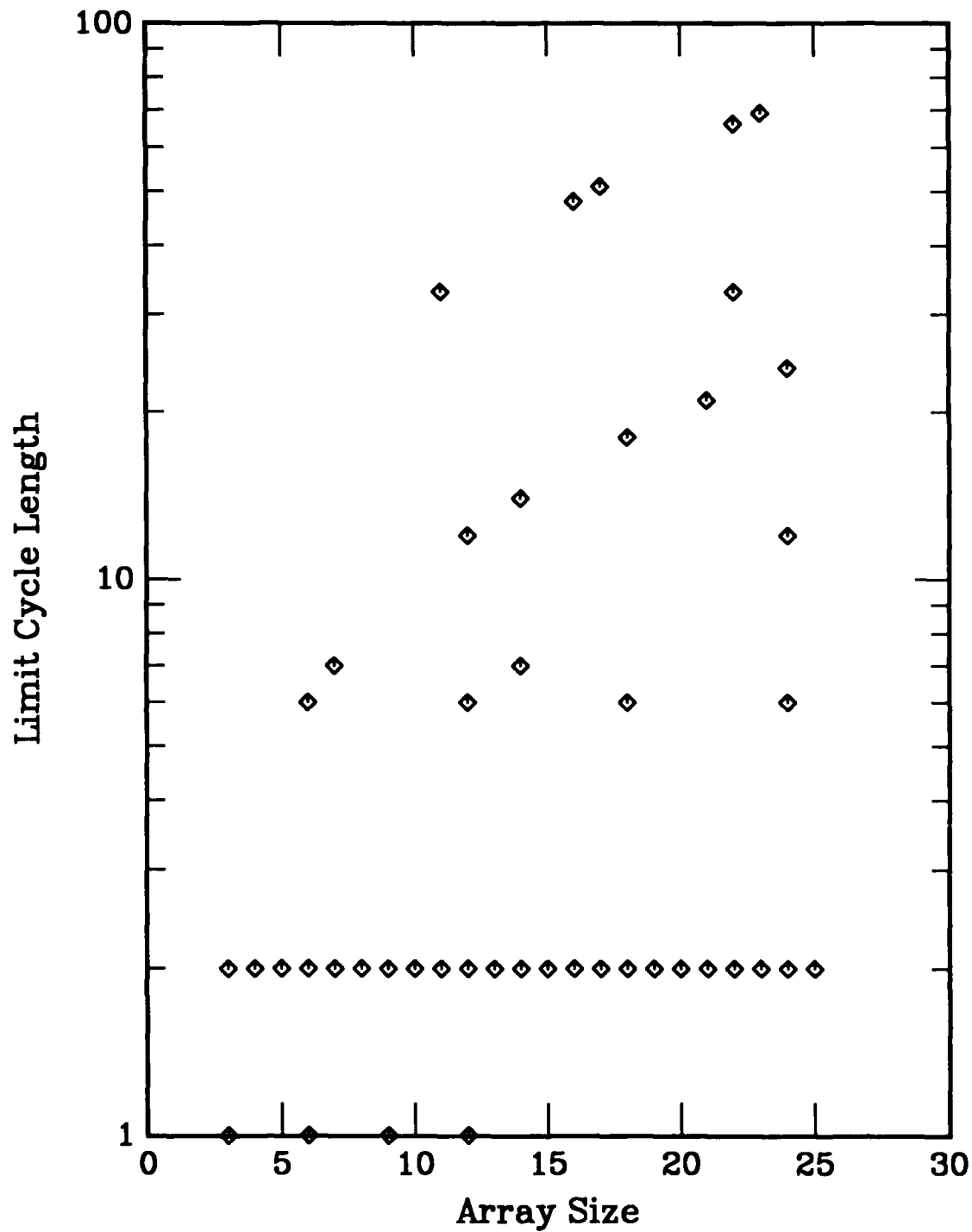


Figure C30. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

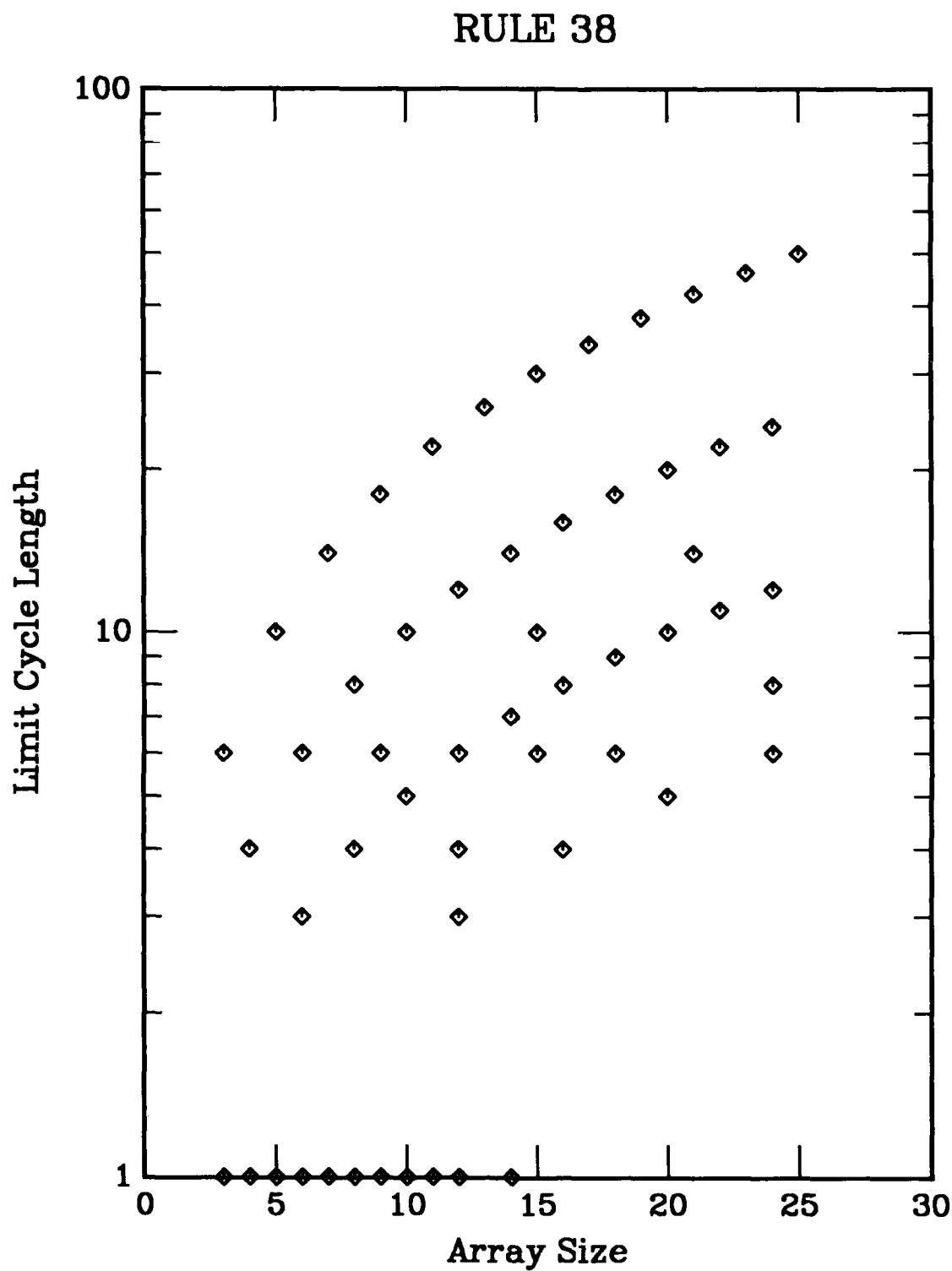


Figure C31. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 40

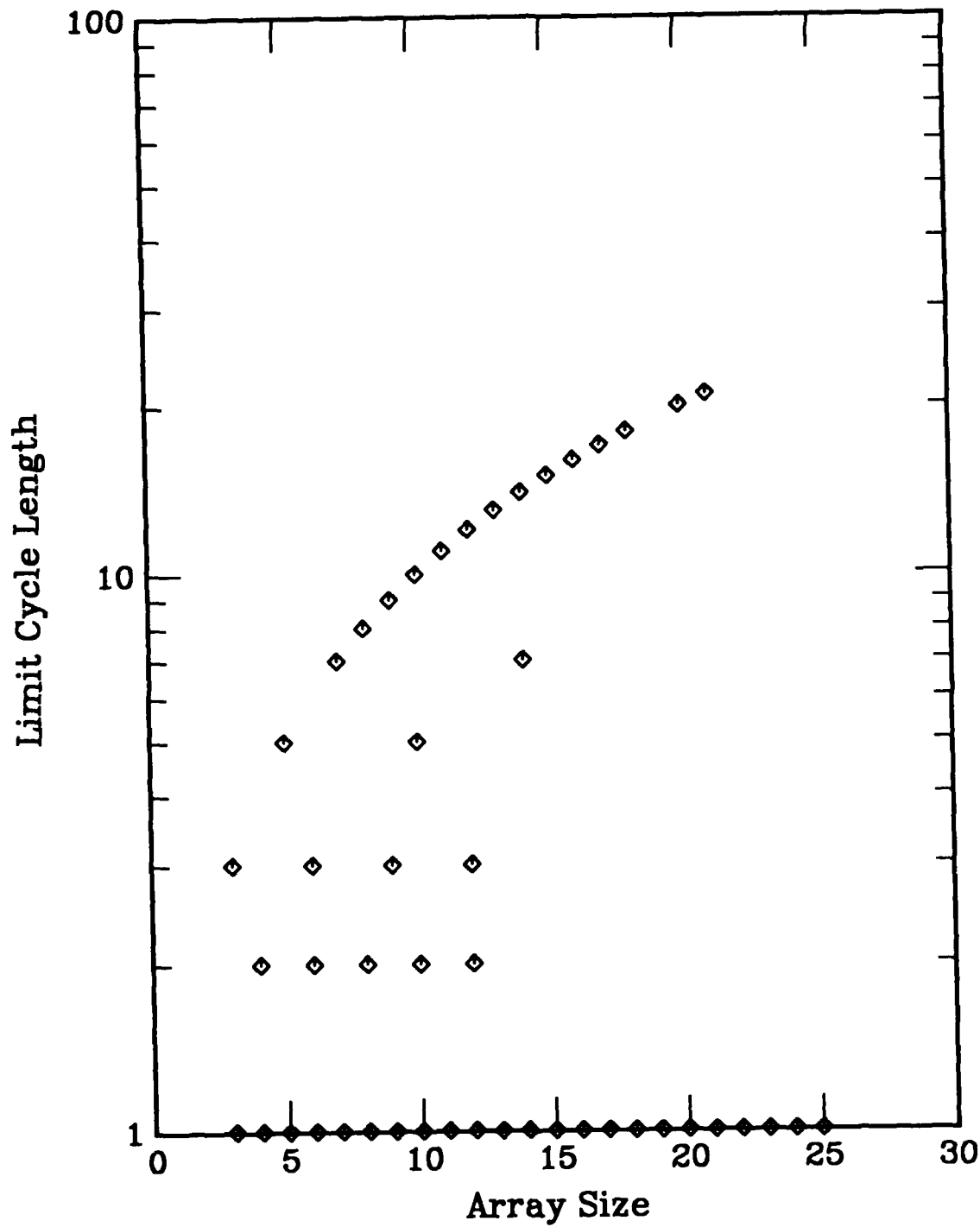


Figure C32. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 41

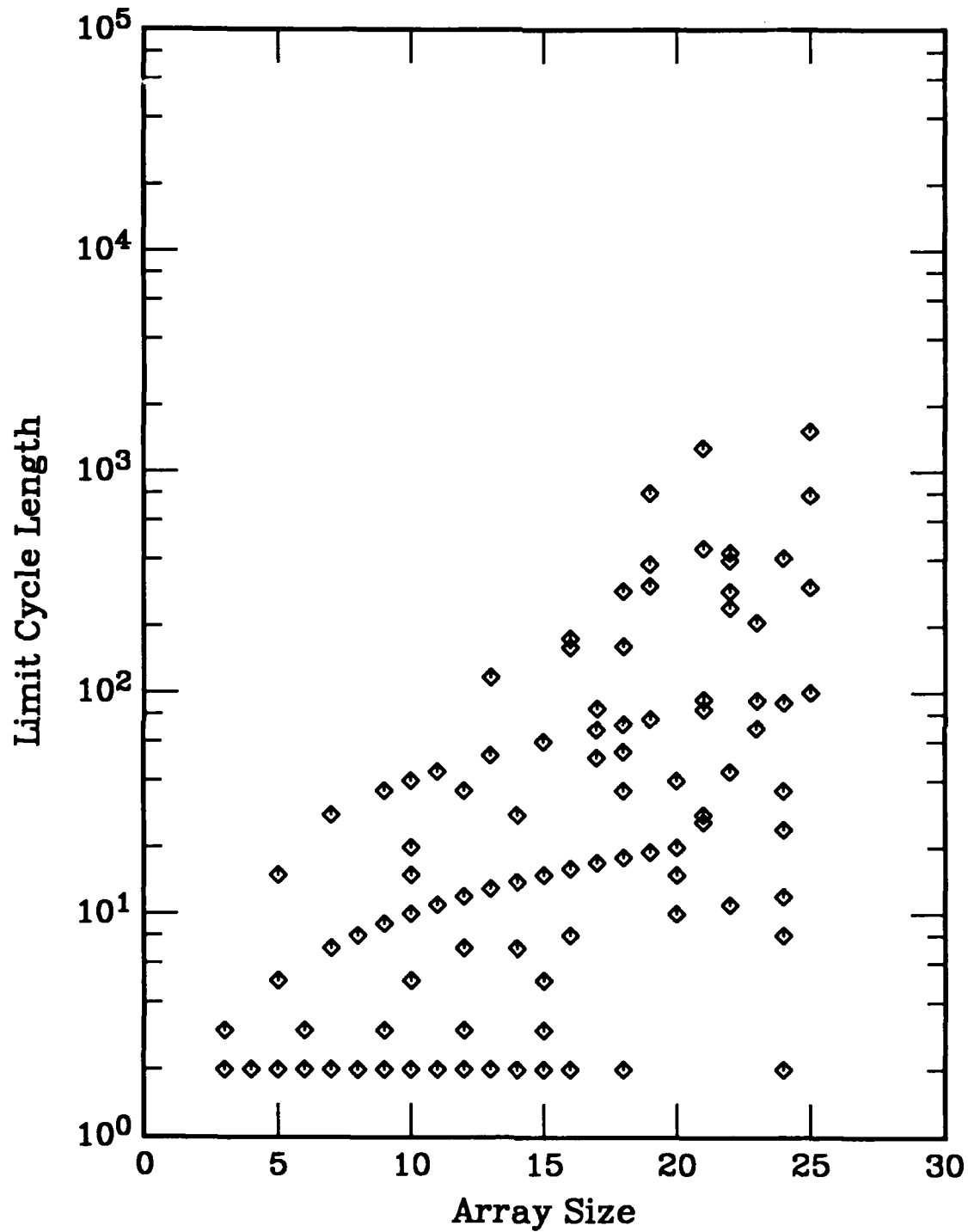


Figure C33. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 42

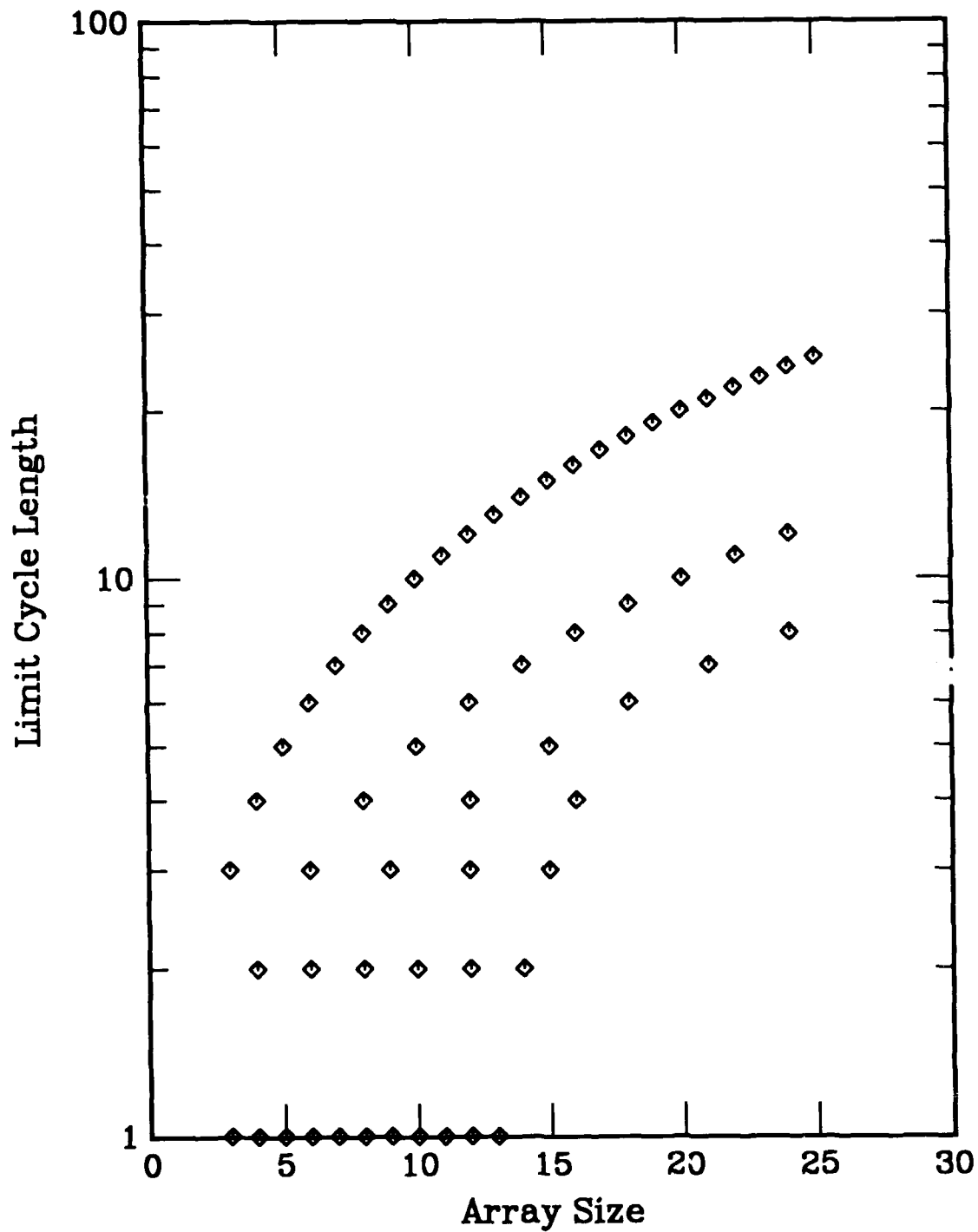


Figure C34. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 43

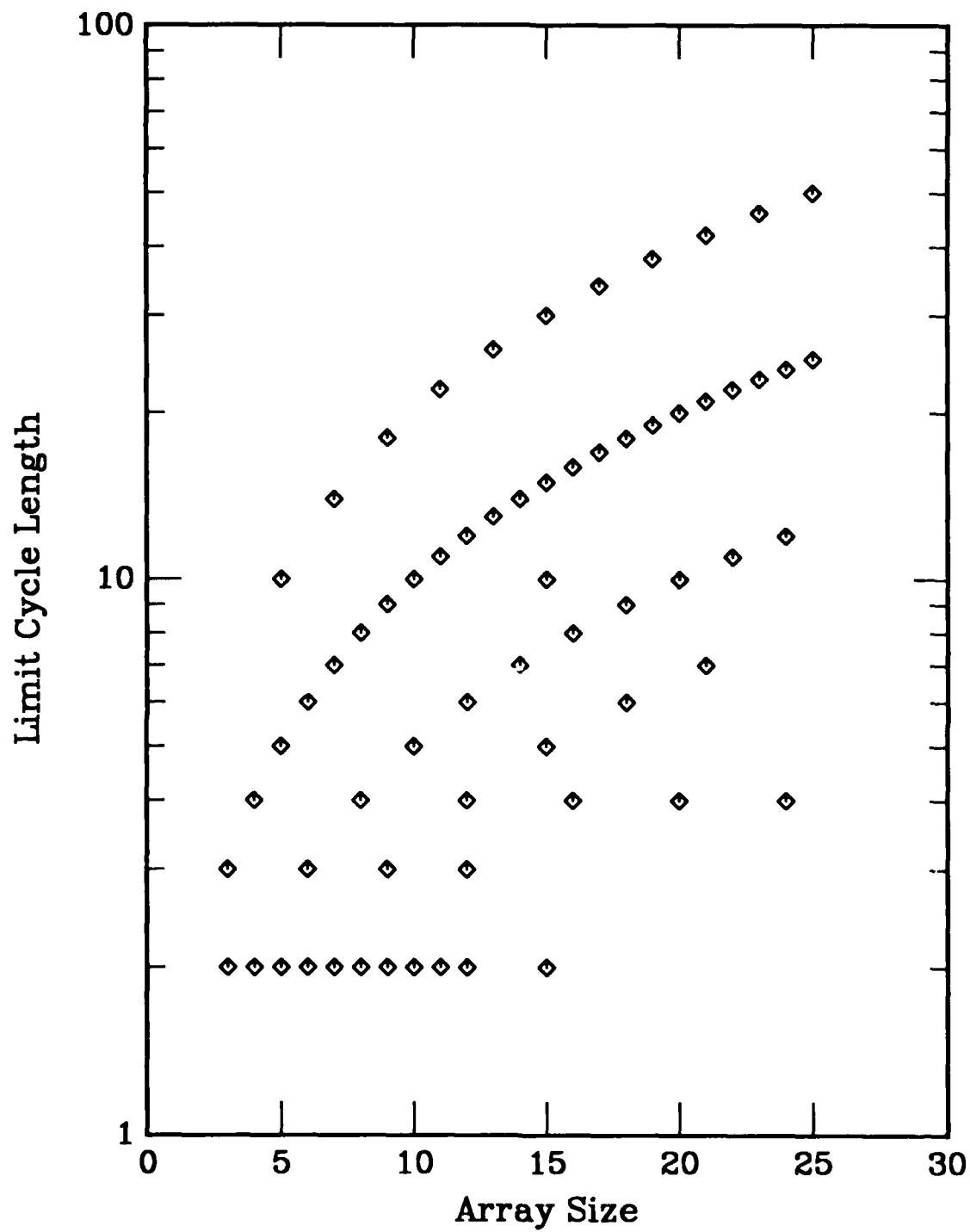


Figure C35. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 44

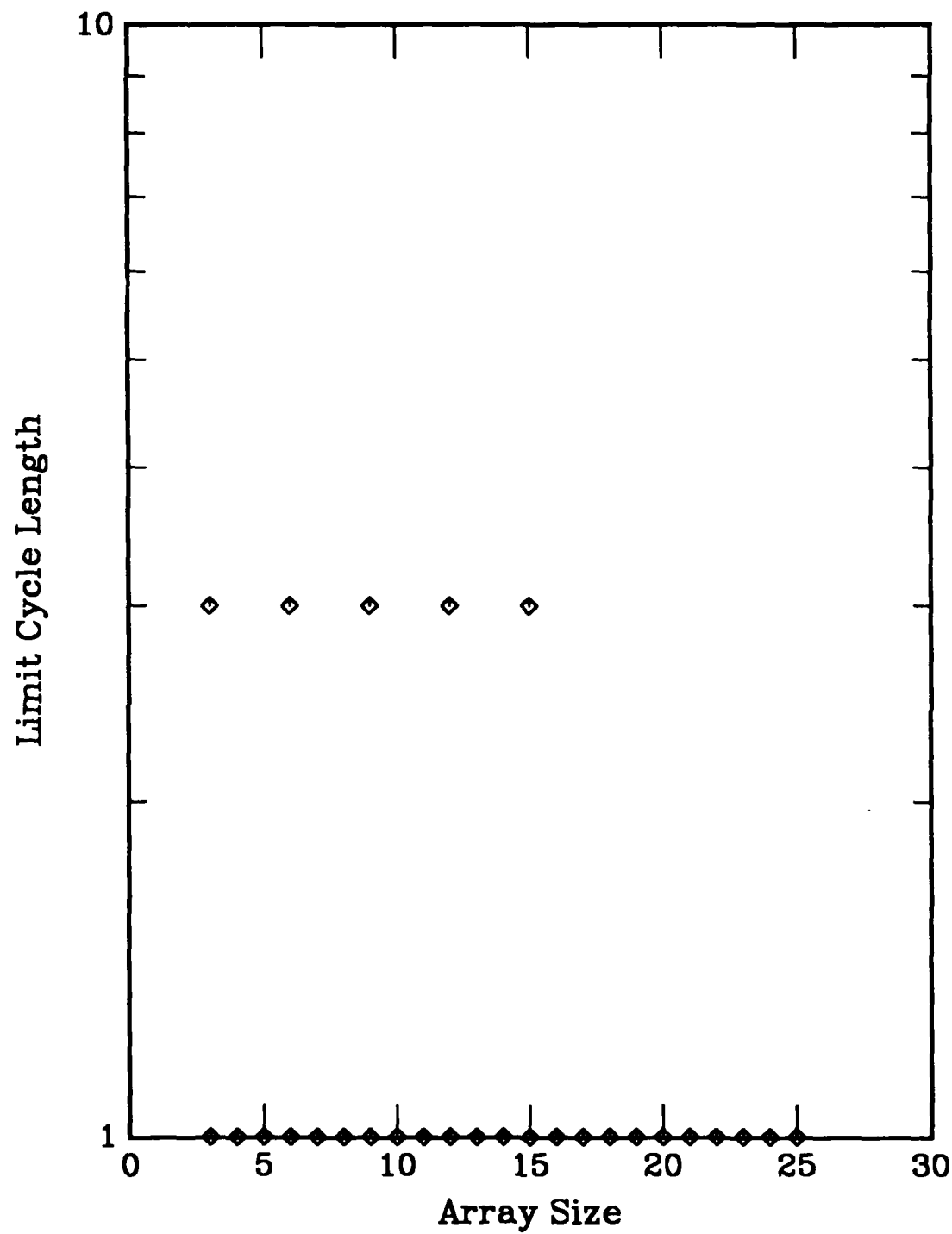


Figure C36. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 45

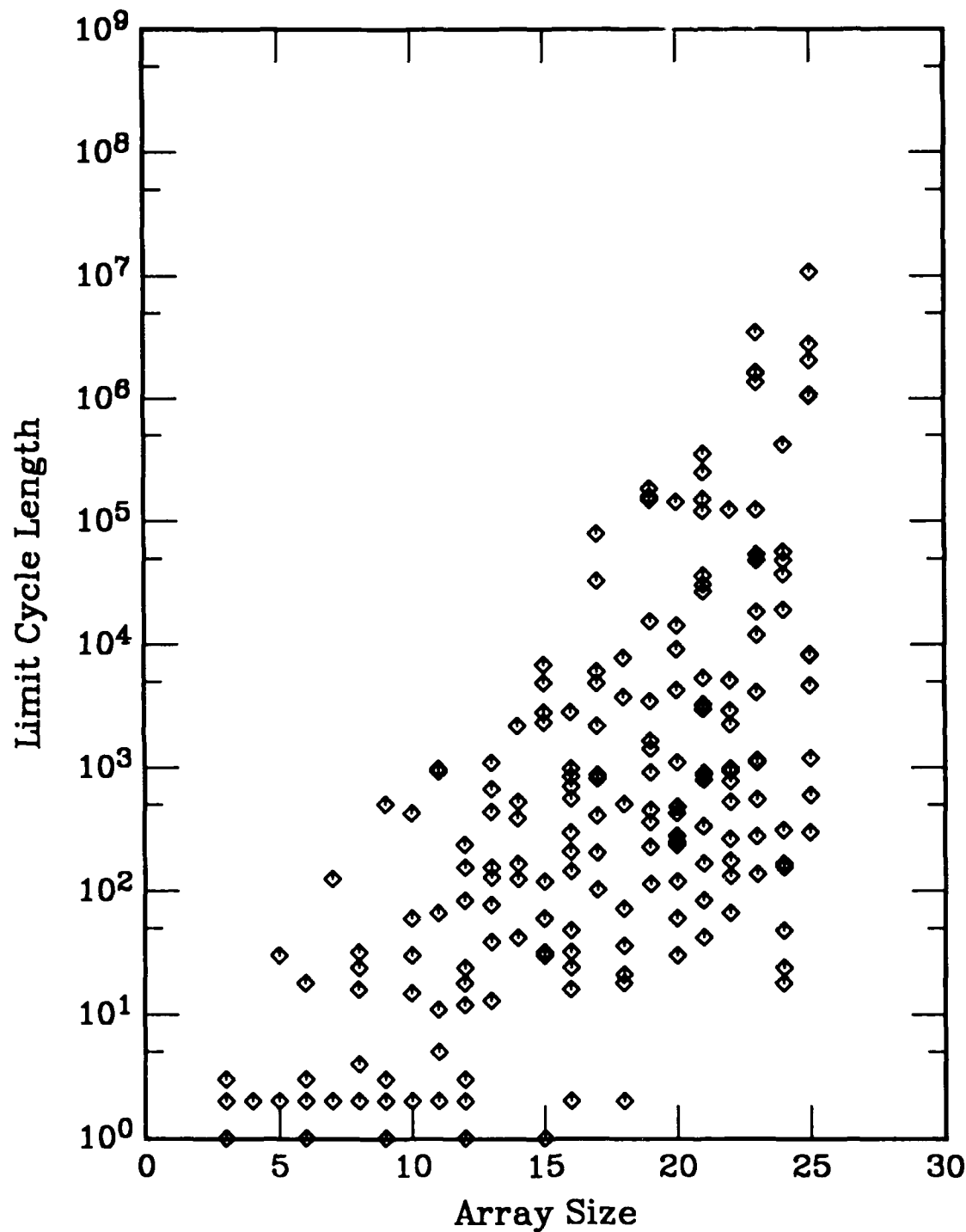


Figure C37. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 46

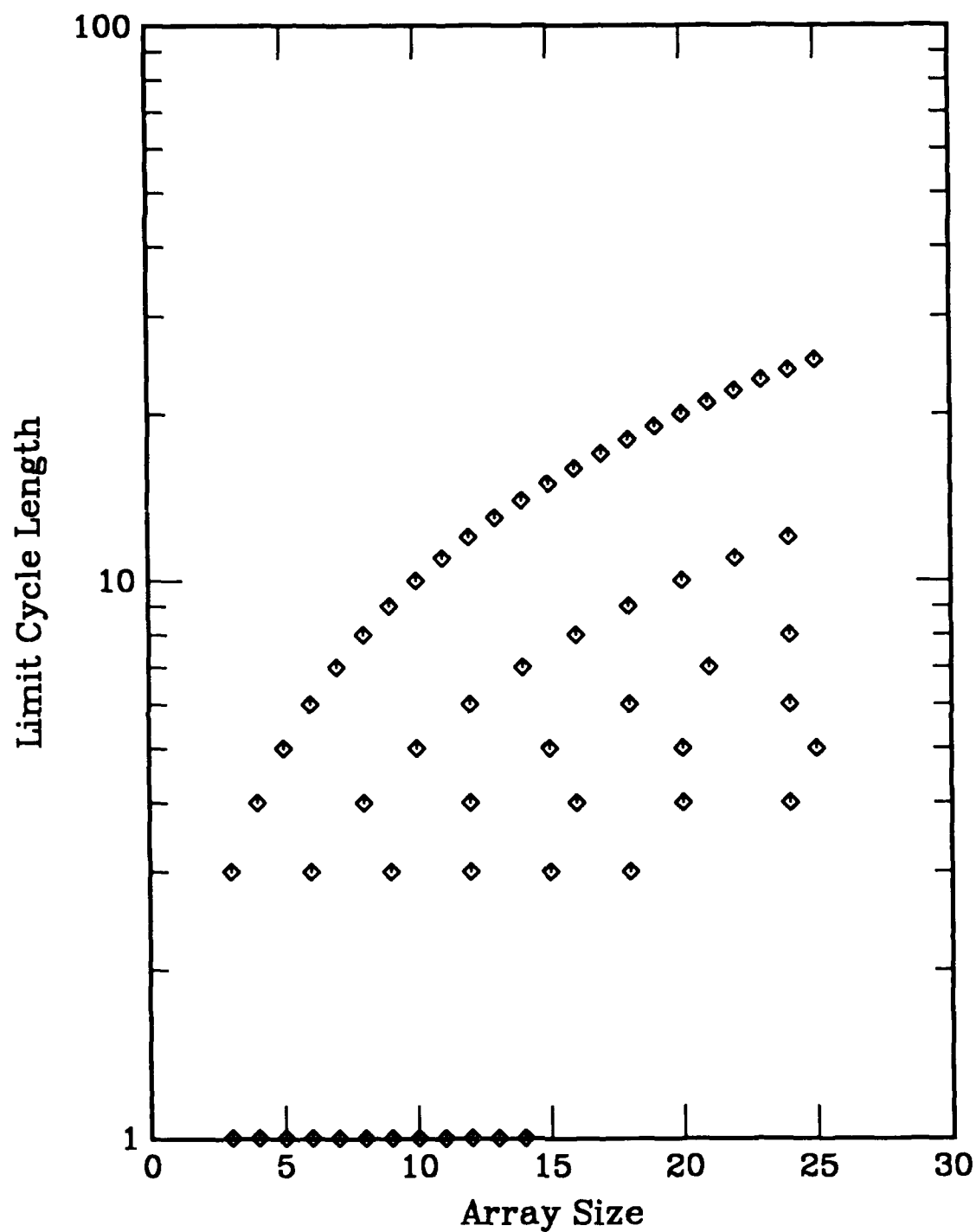


Figure C38. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

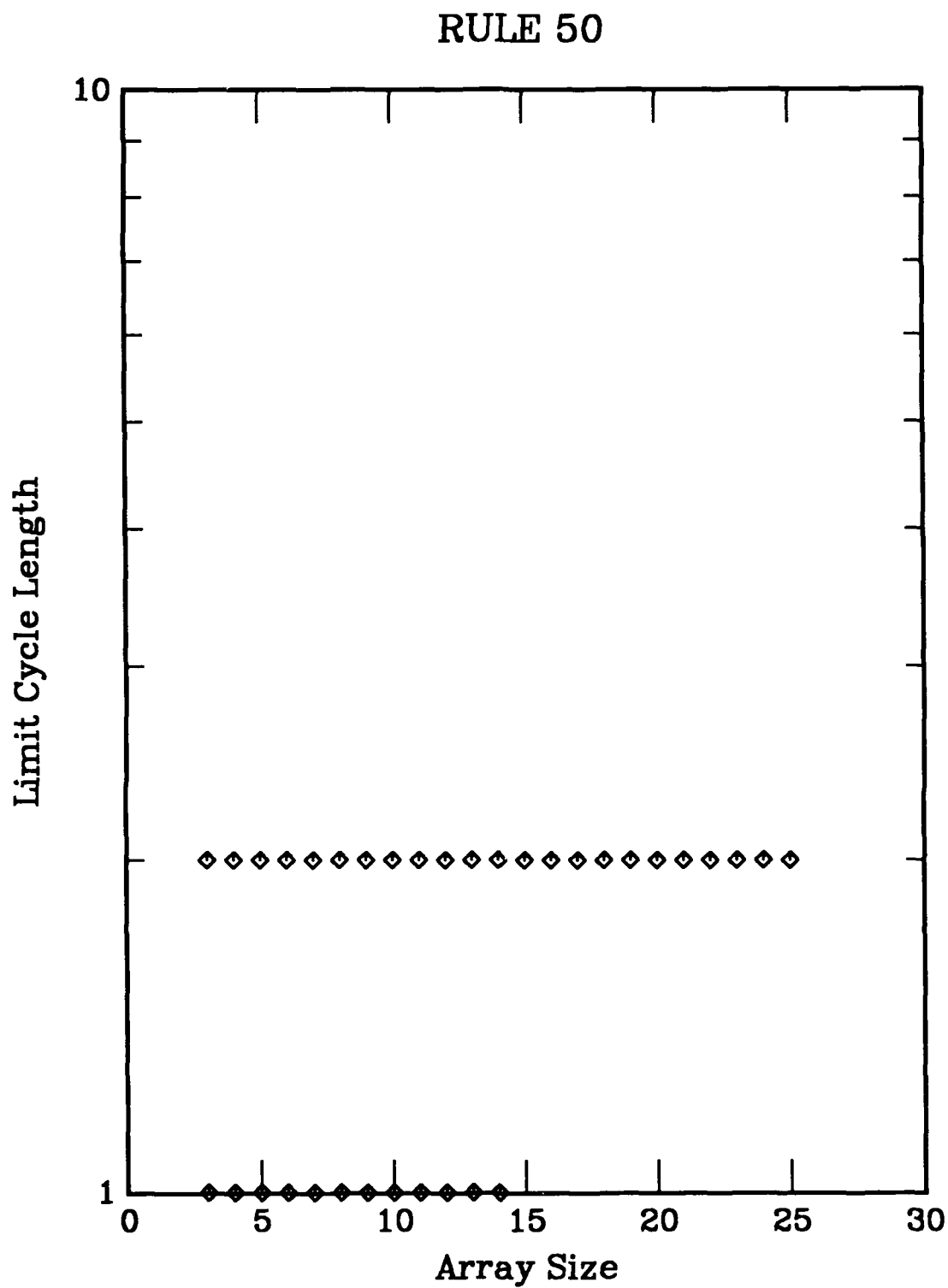


Figure C39. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 51

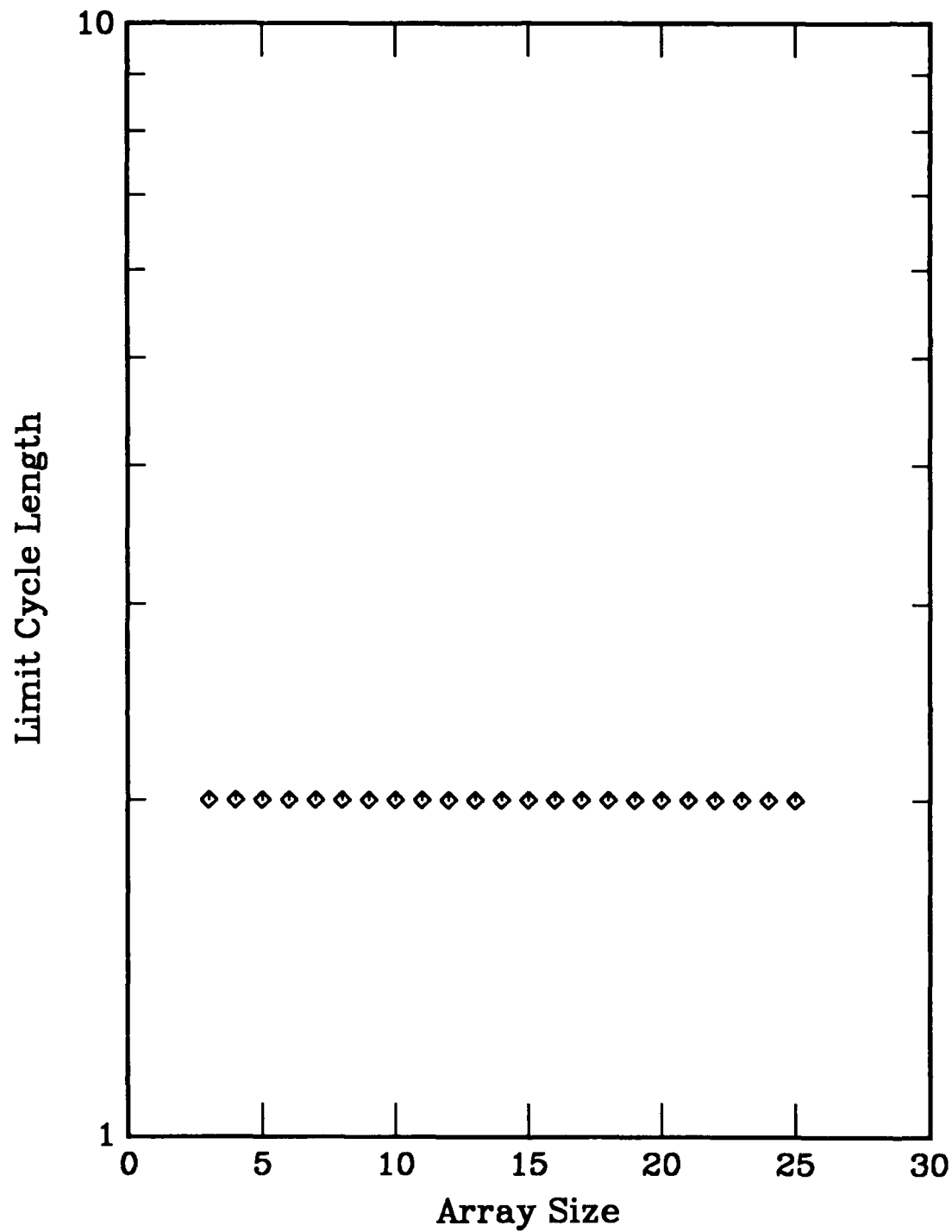


Figure C40. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 54

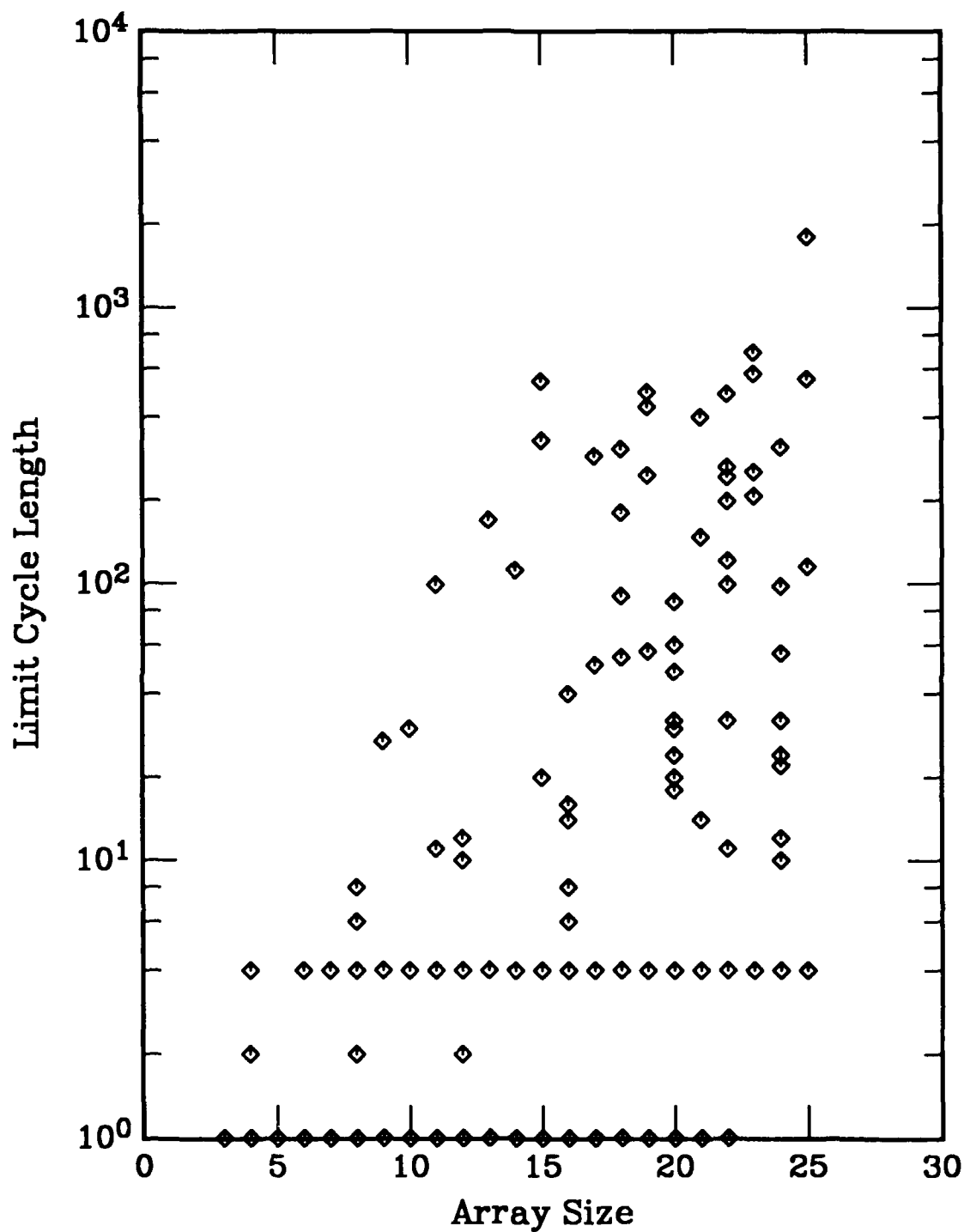


Figure C41. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 56

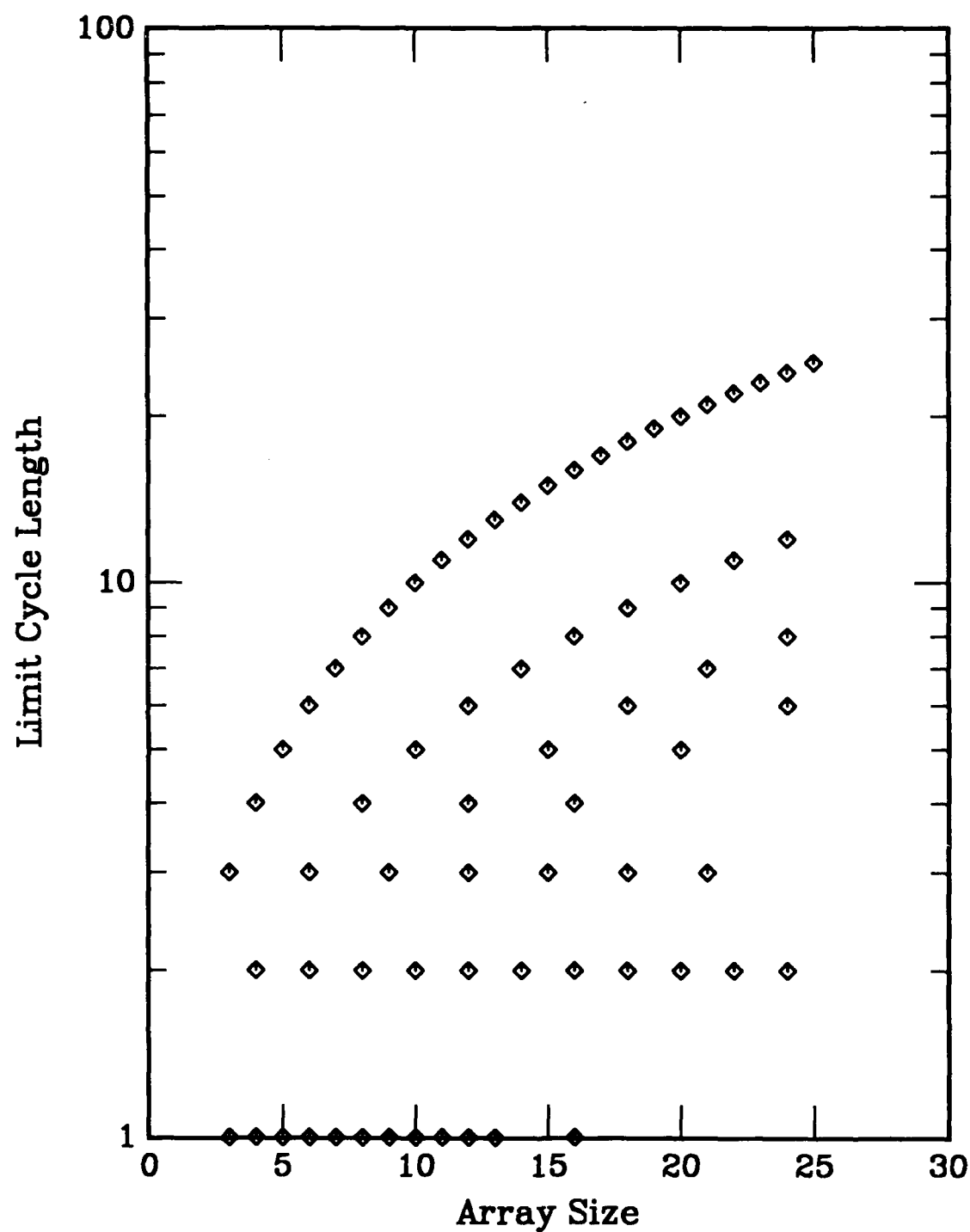


Figure C42. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 57

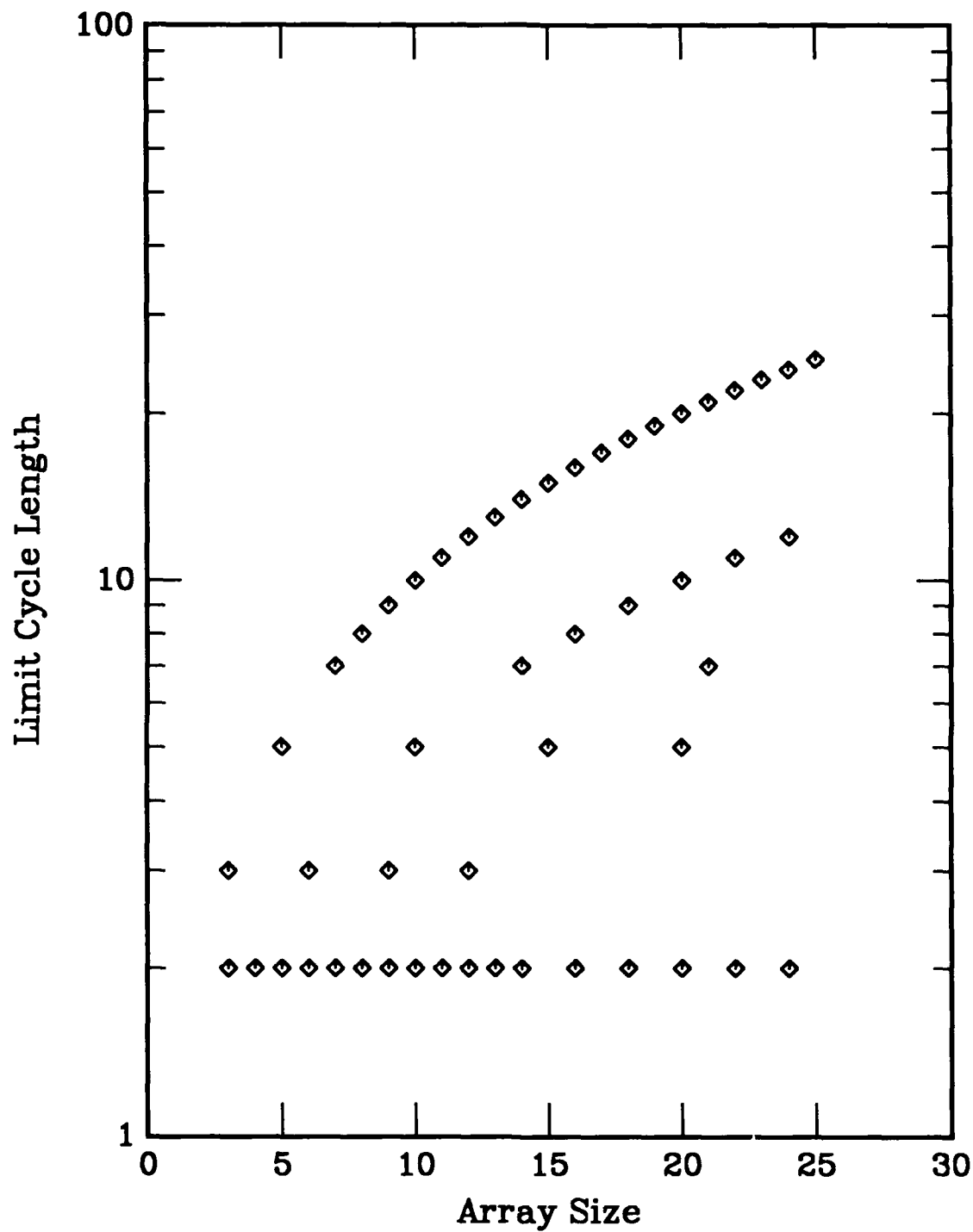


Figure C43. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 58

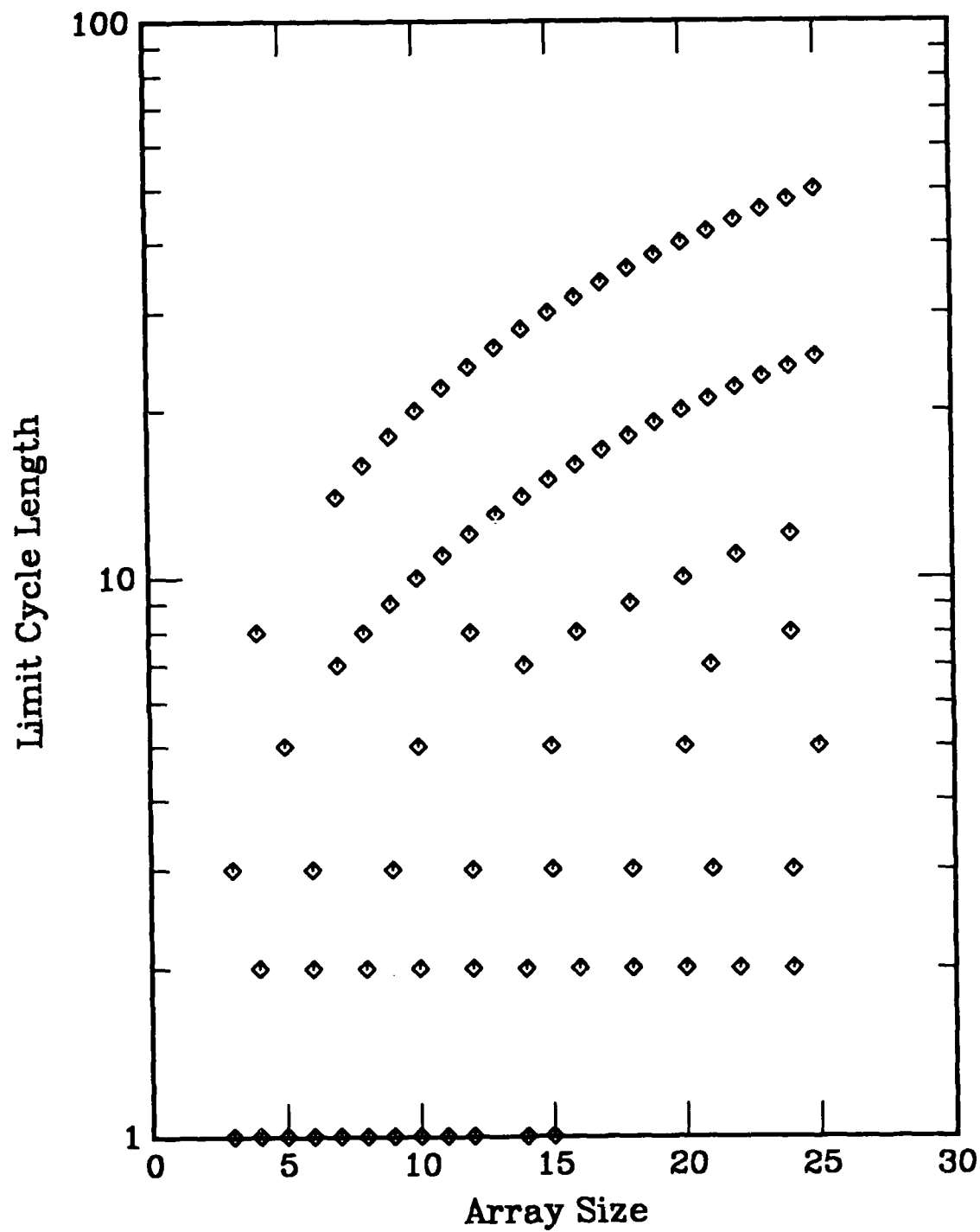


Figure C44. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 60

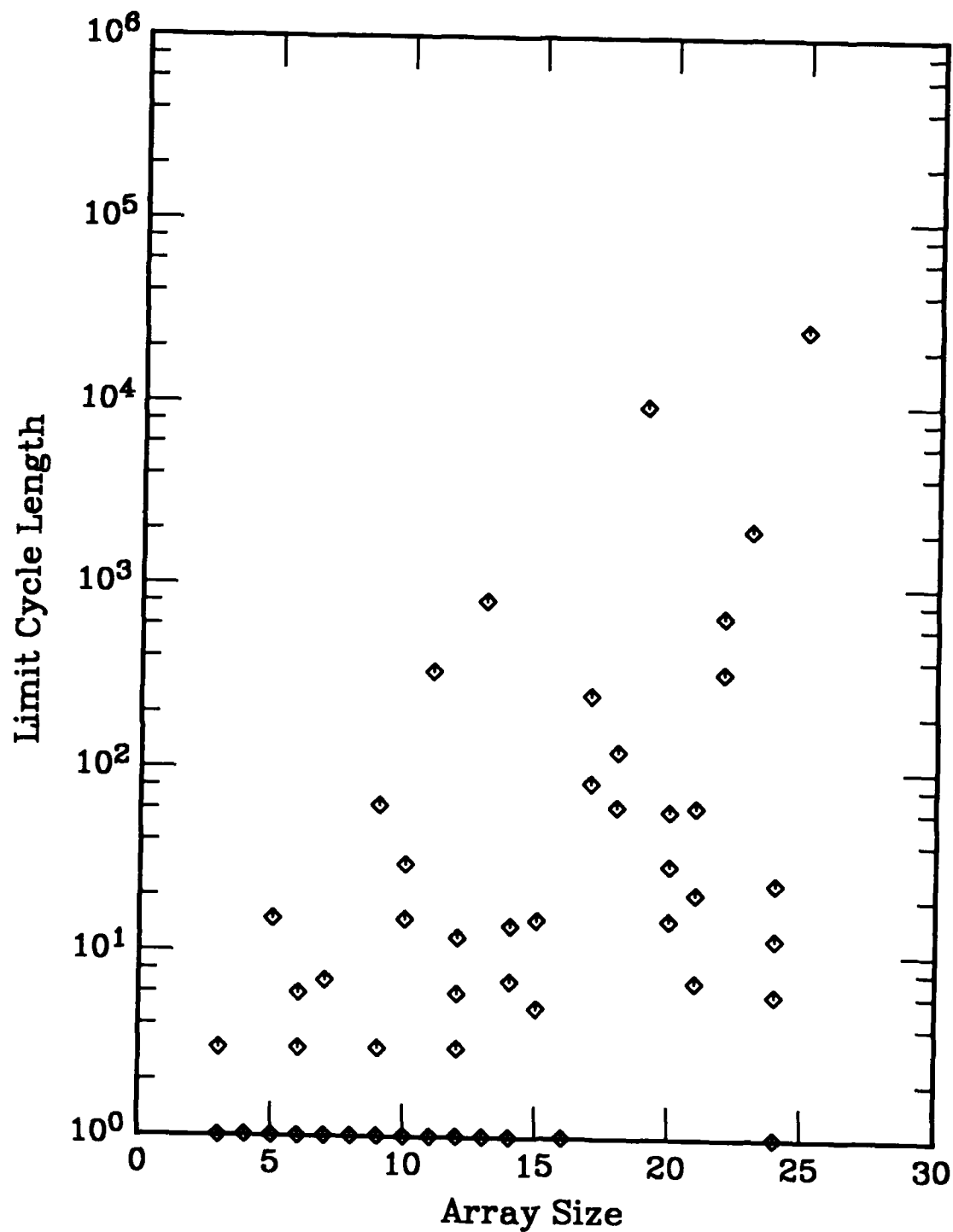


Figure C45. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 62

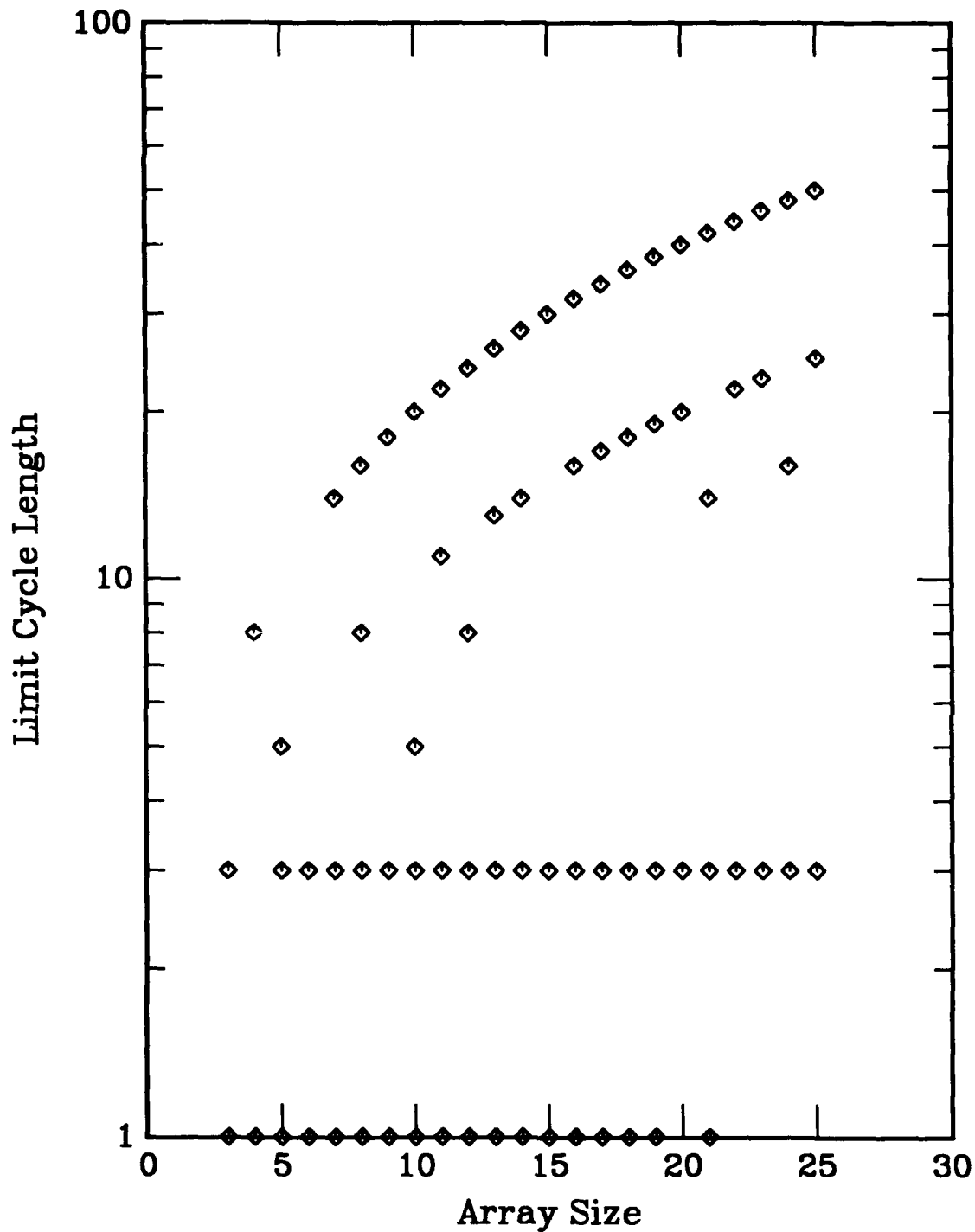


Figure C46. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 72

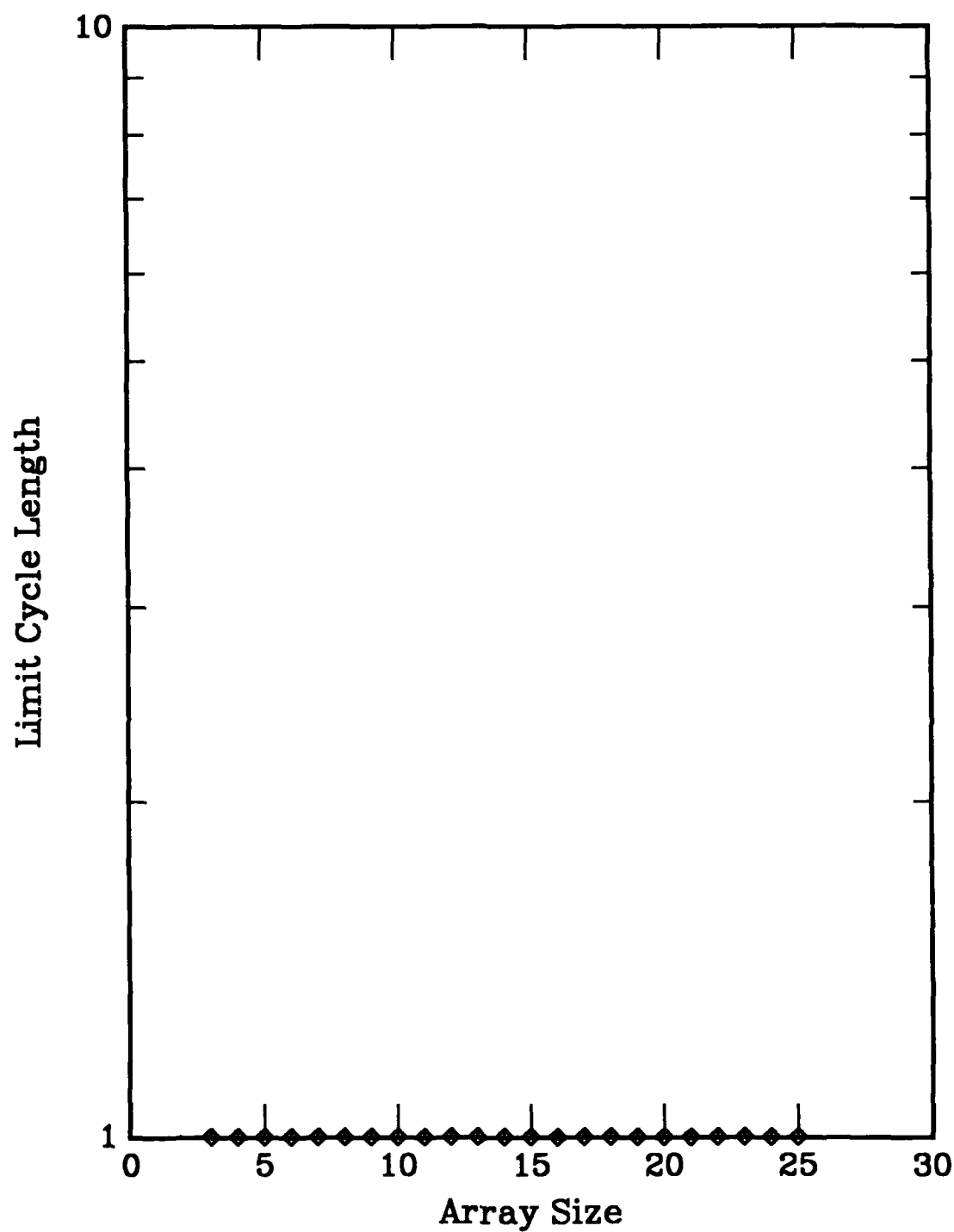


Figure C47. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 73

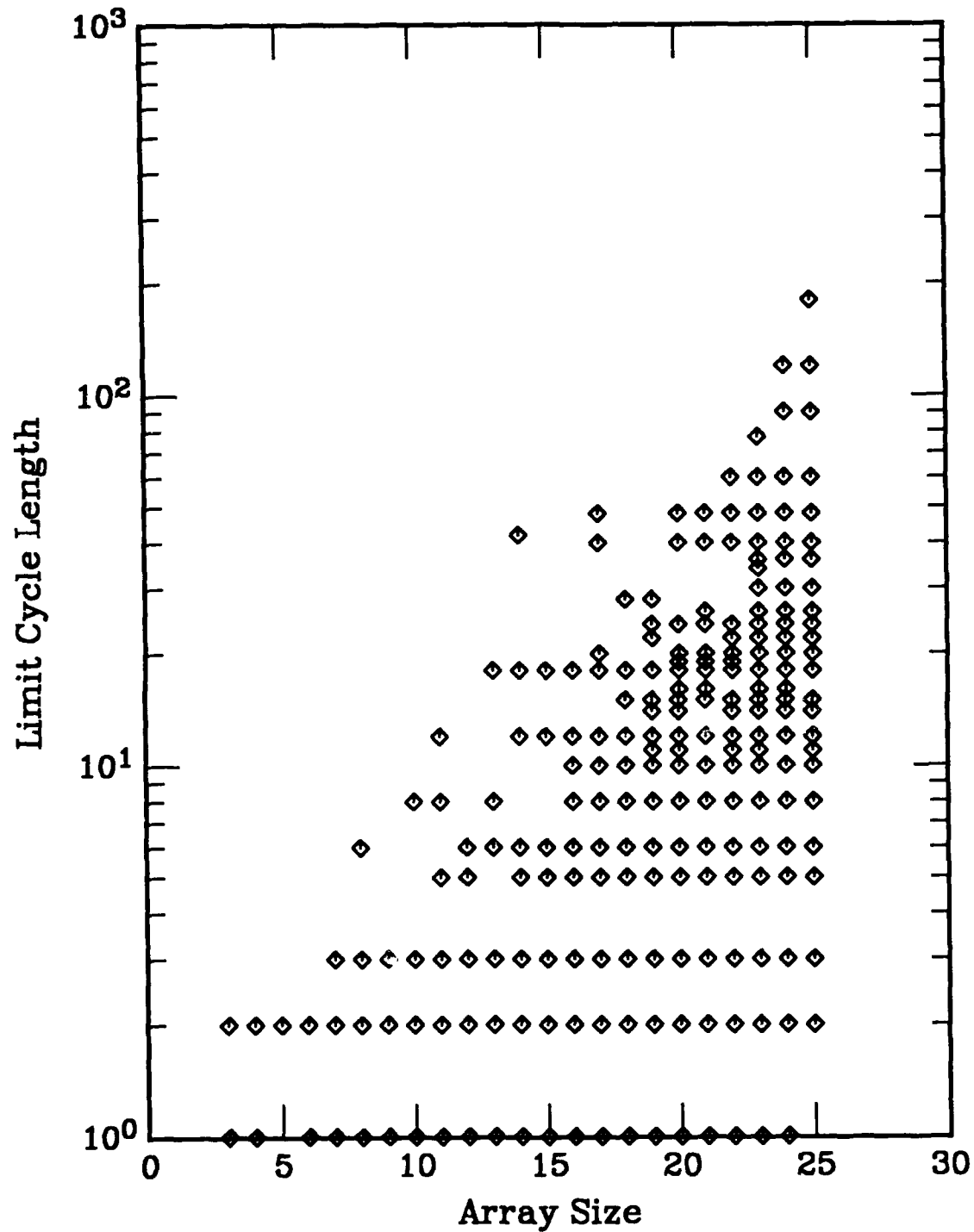


Figure C48. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 74

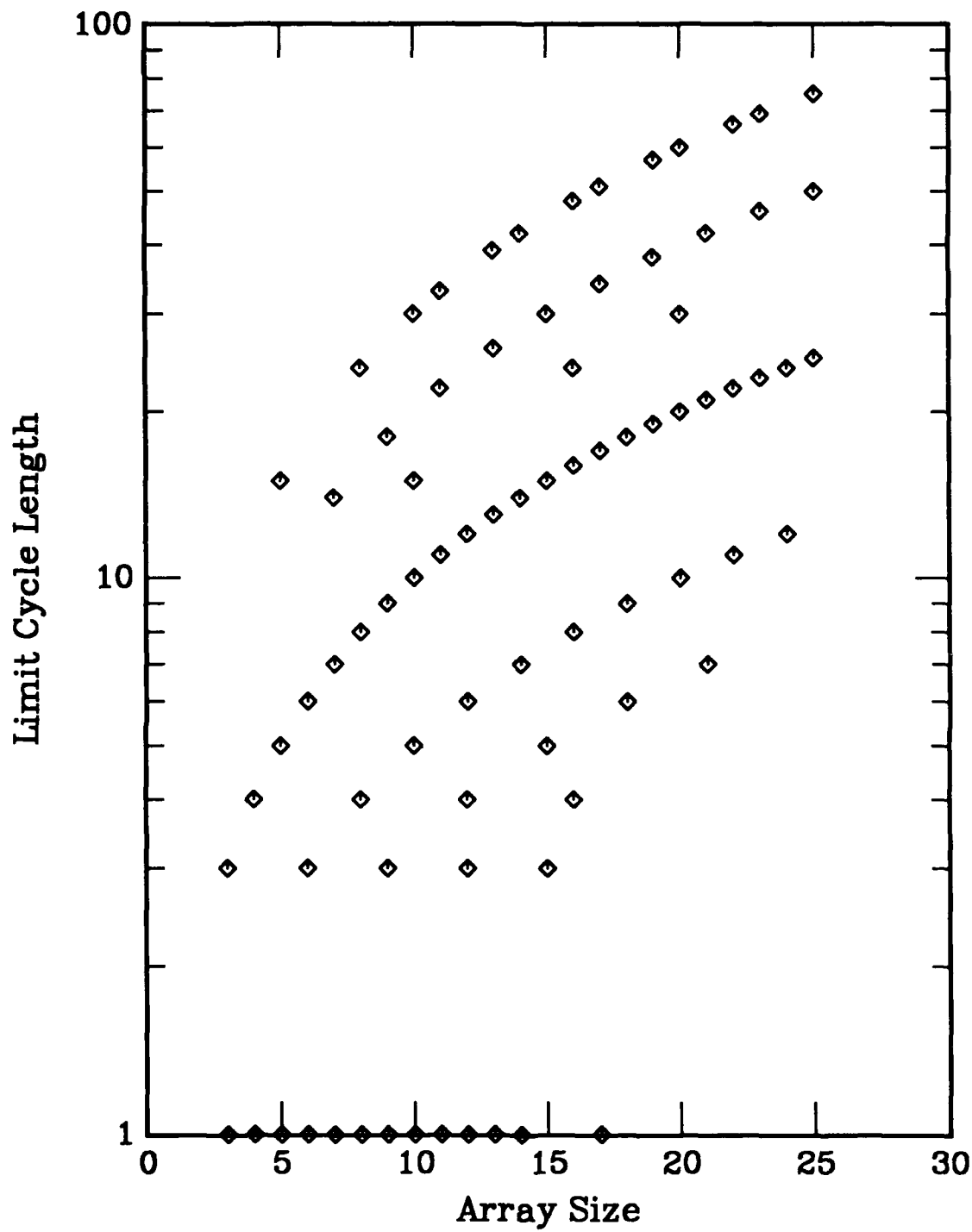


Figure C49. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 76

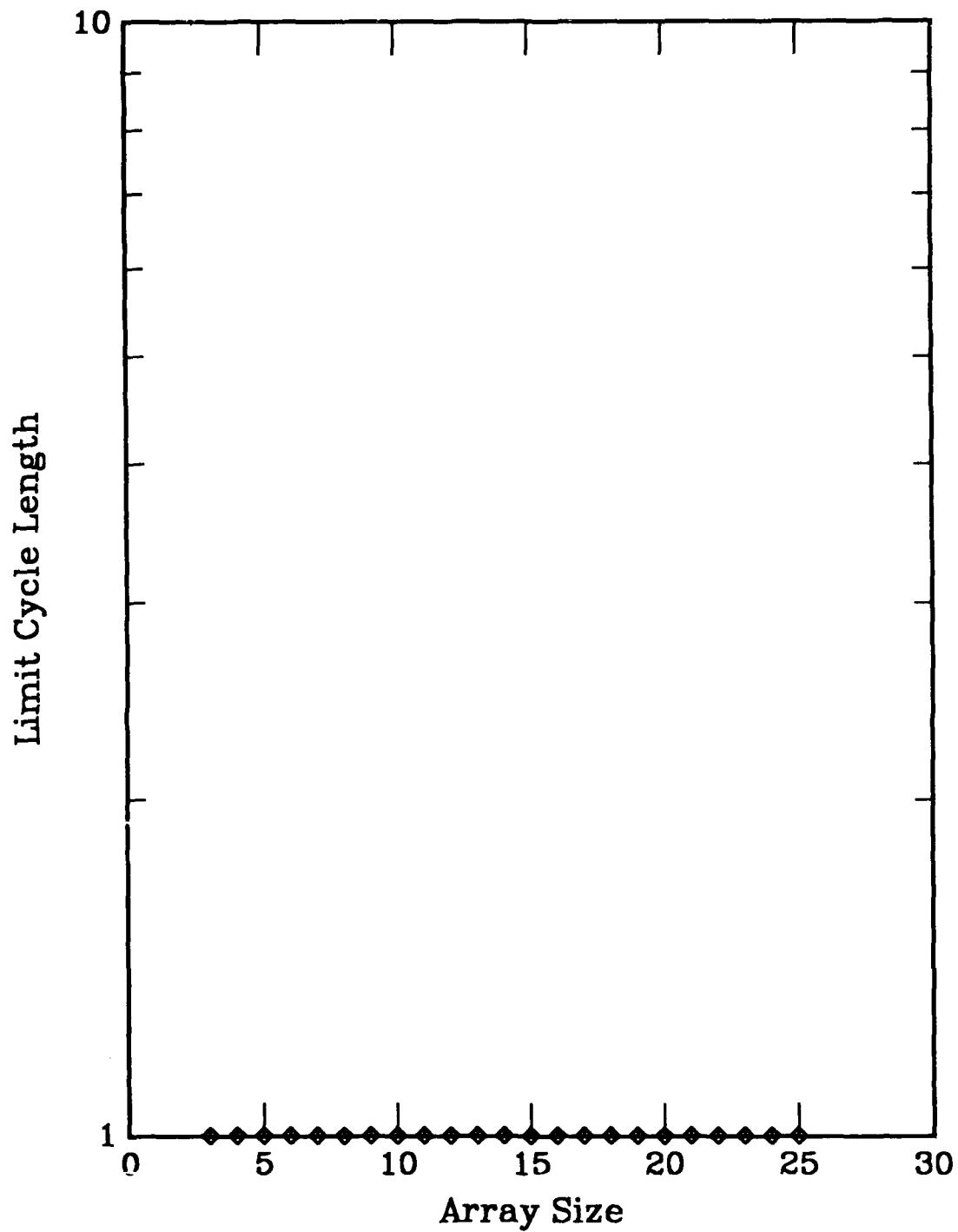


Figure C50. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

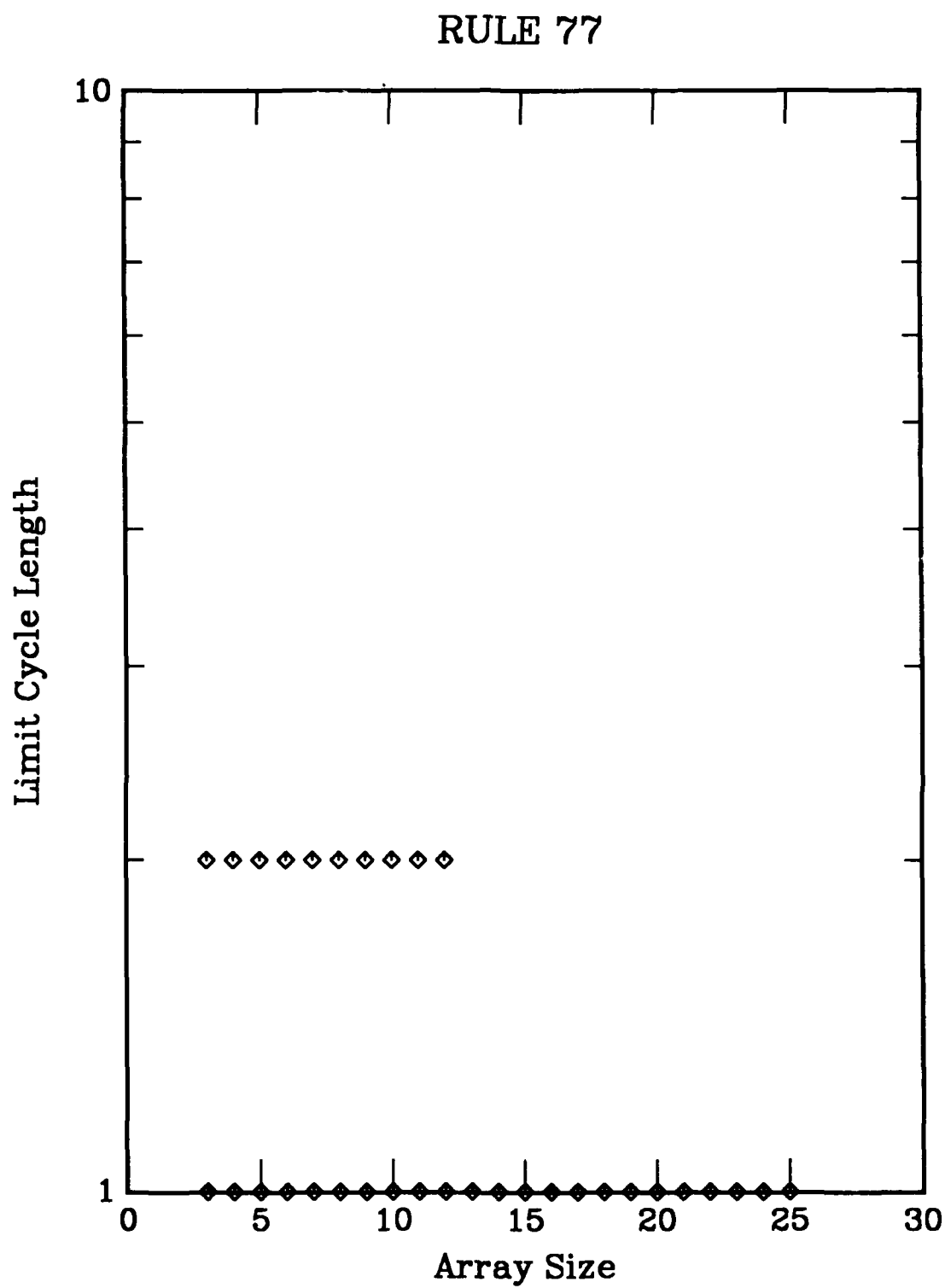


Figure C51. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 78

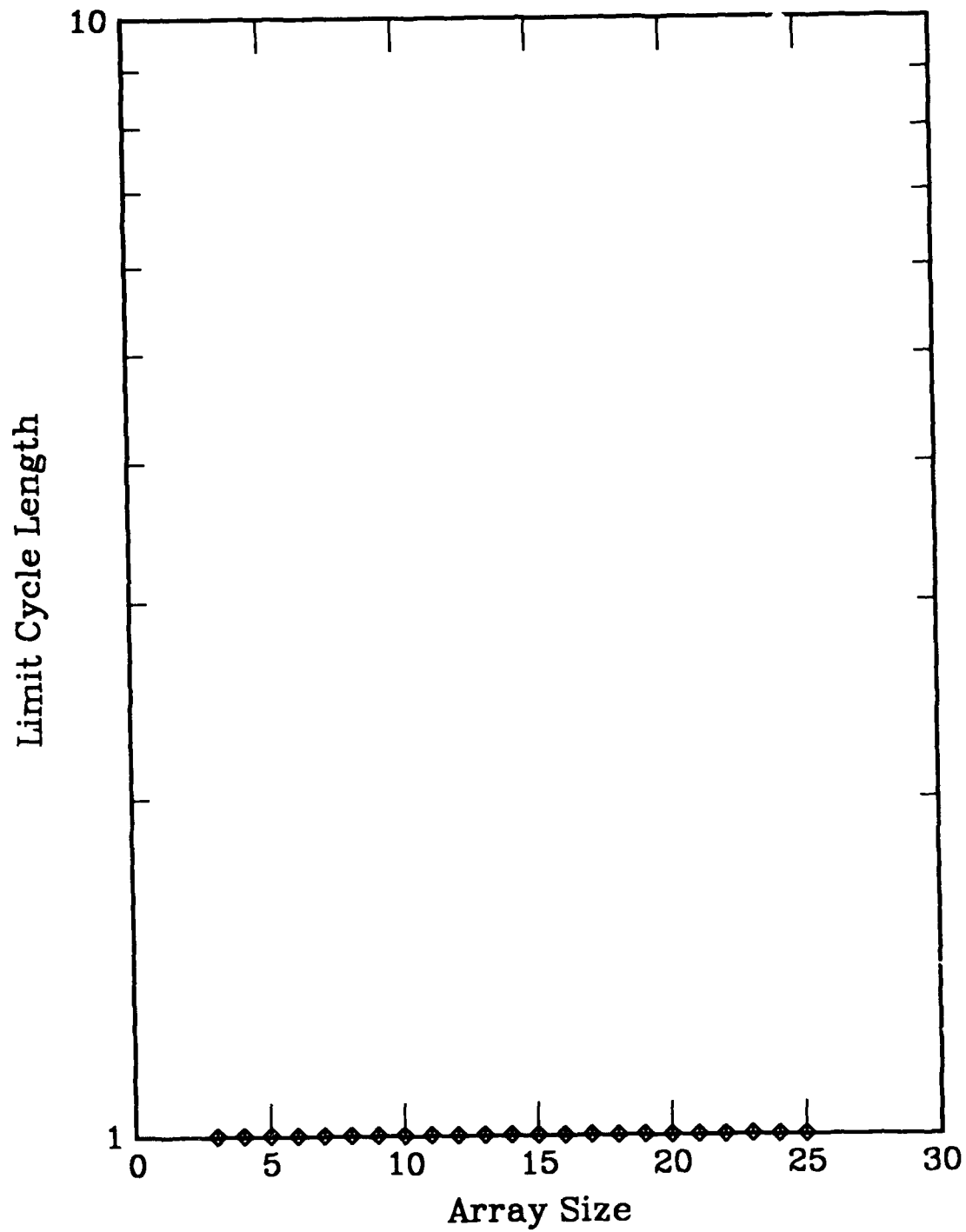


Figure C52. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

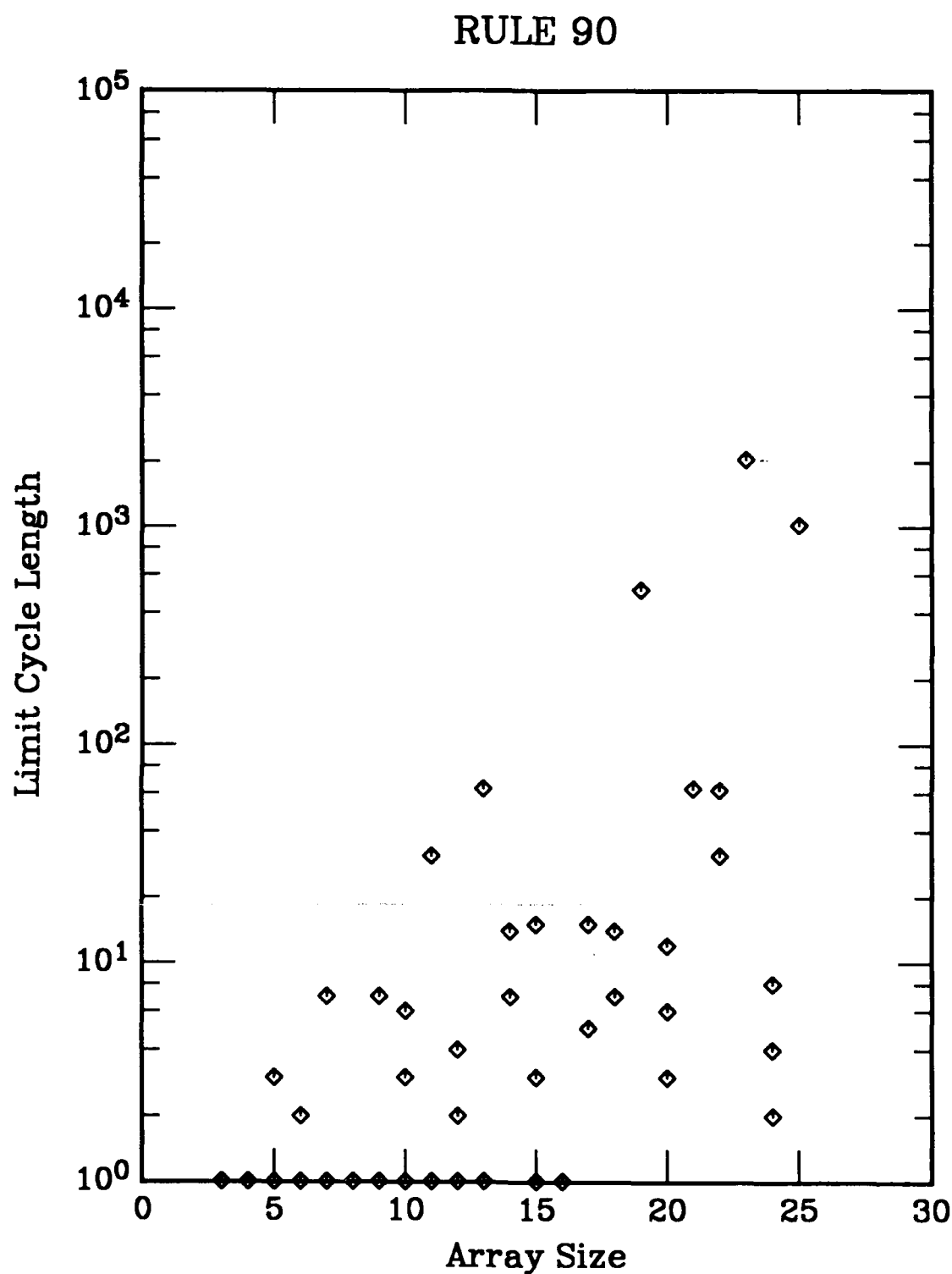


Figure C53. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 94

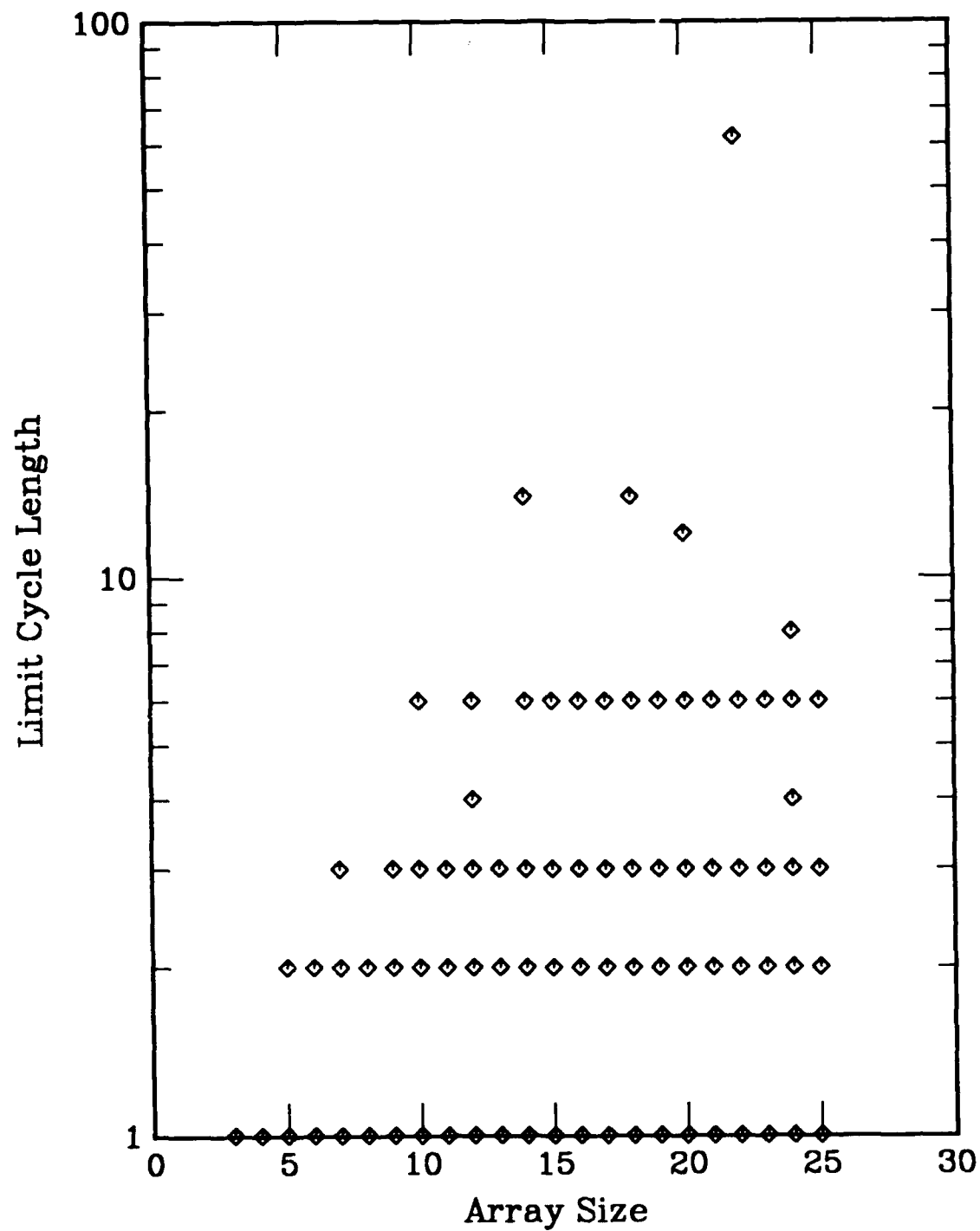


Figure C54. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

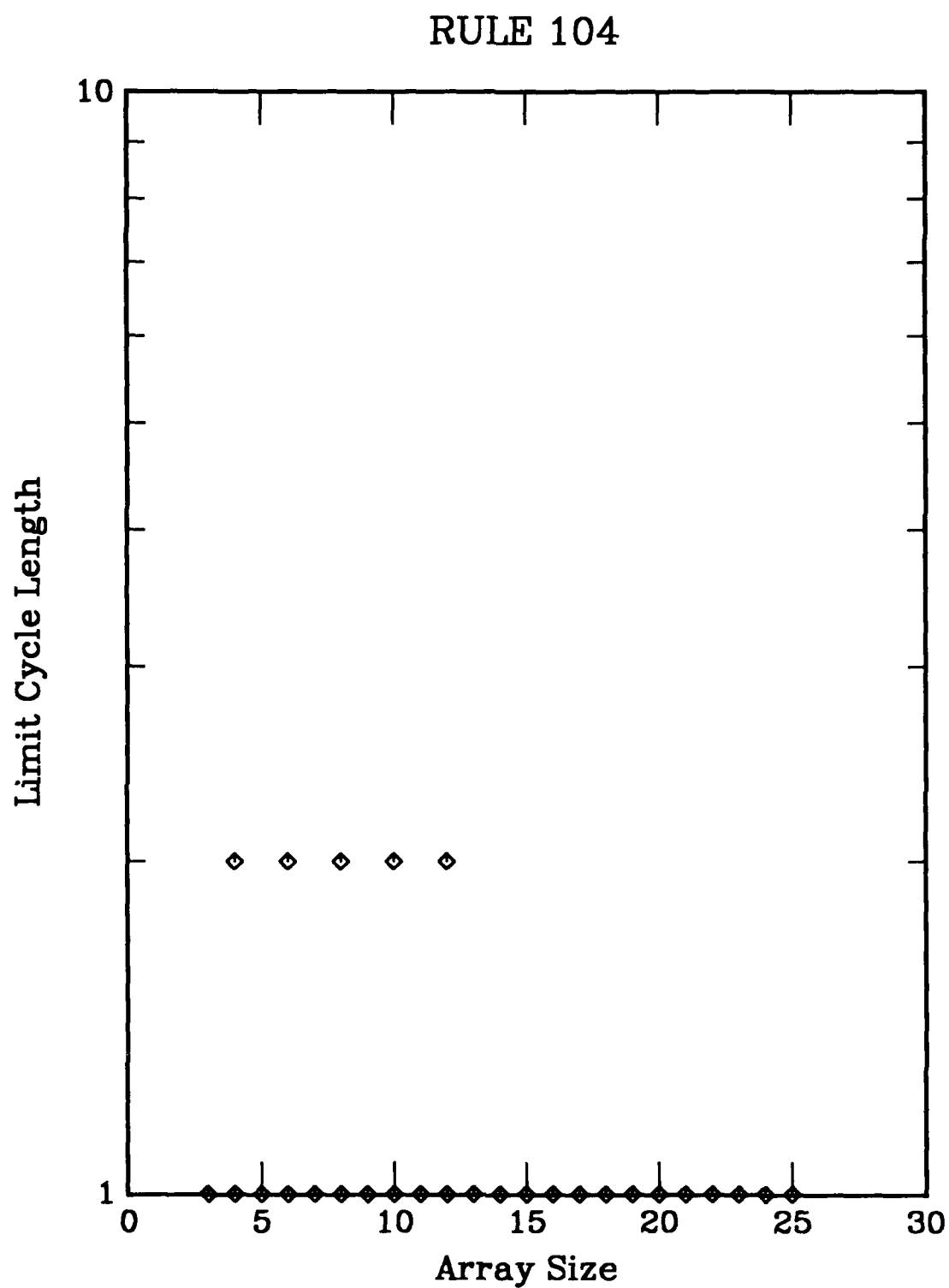


Figure C55. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 105

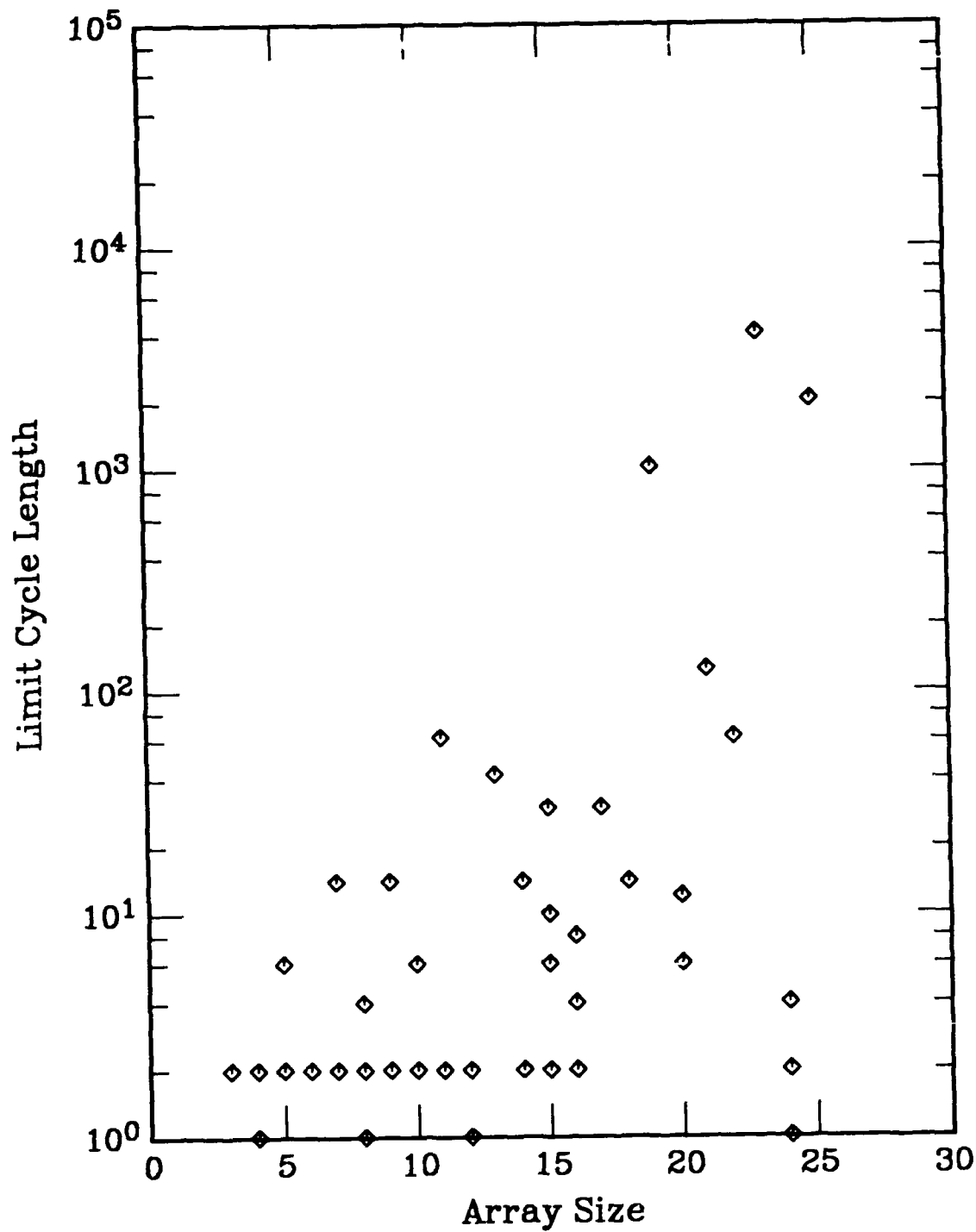


Figure C56. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

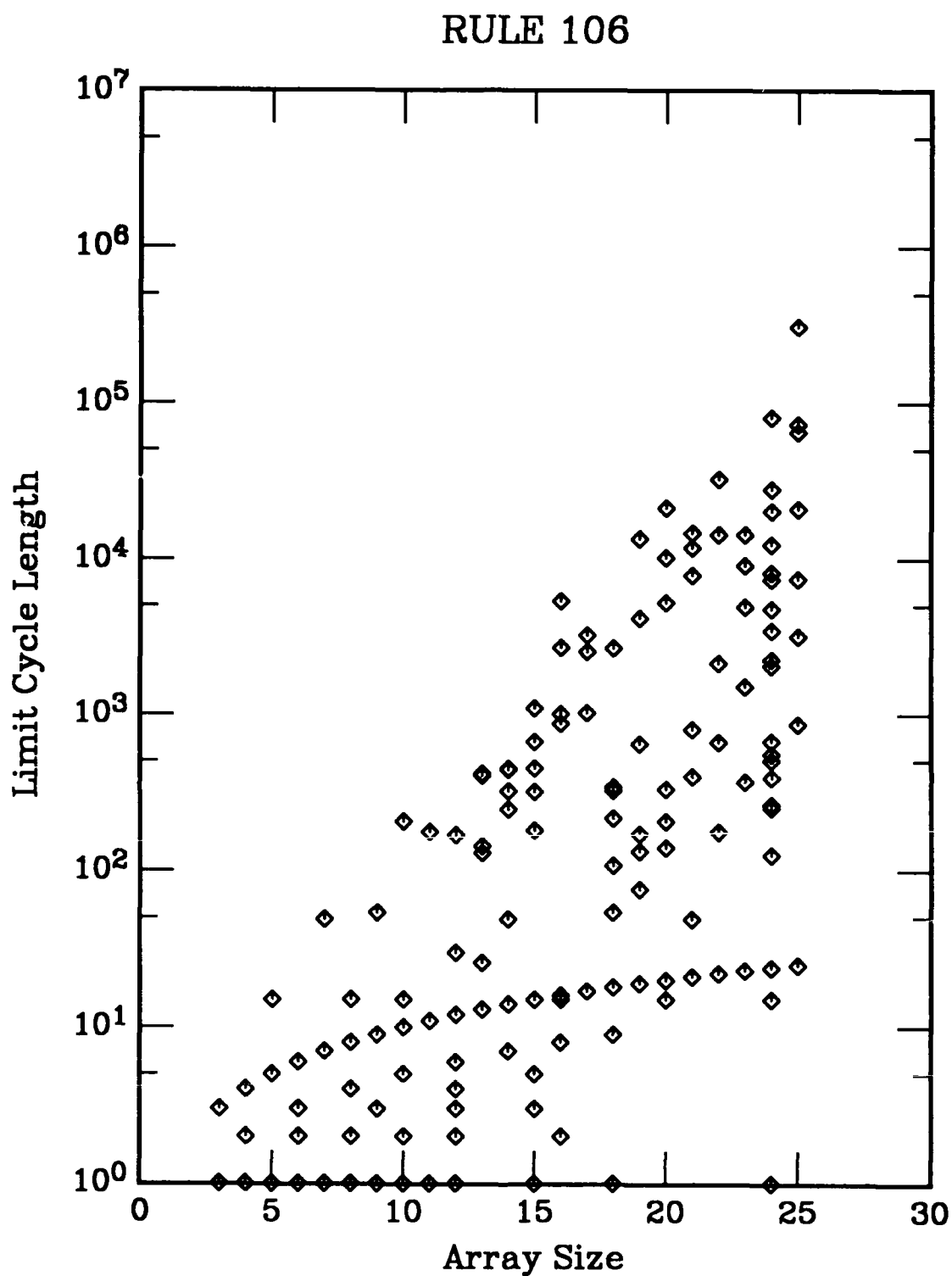


Figure C57. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 108

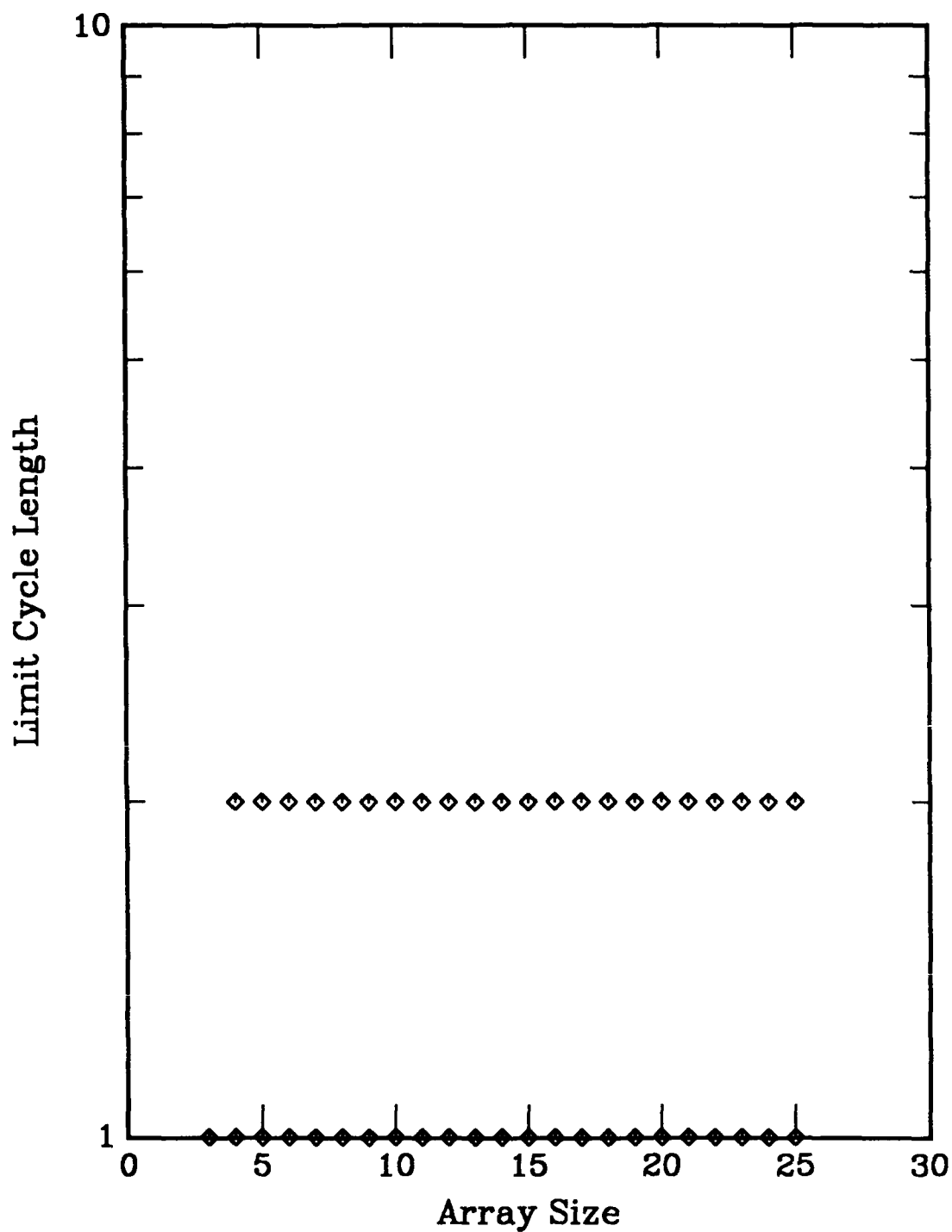


Figure C58. Limit cycle length vs array size for a one-dimensional, nearest neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 110

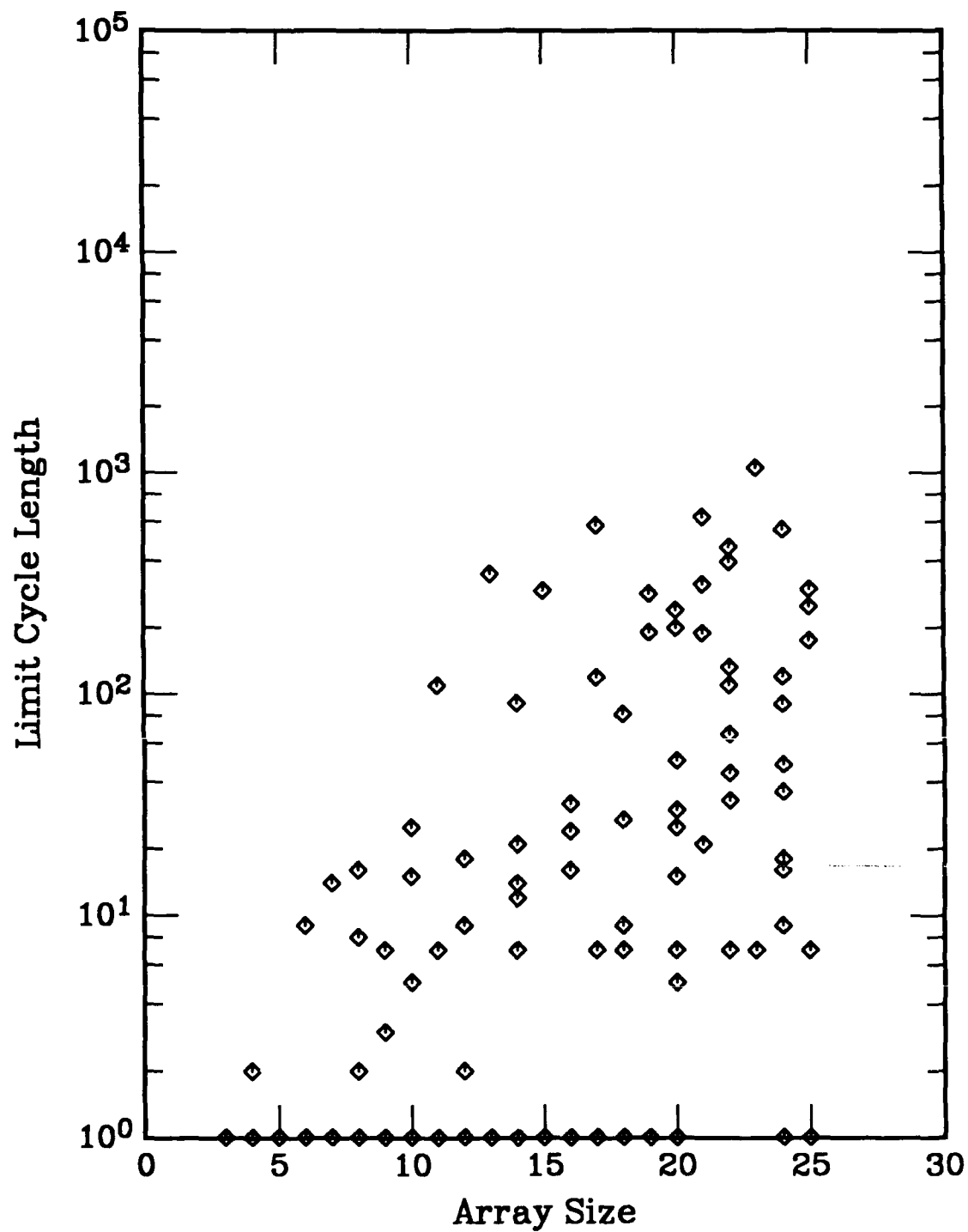


Figure C59. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 122

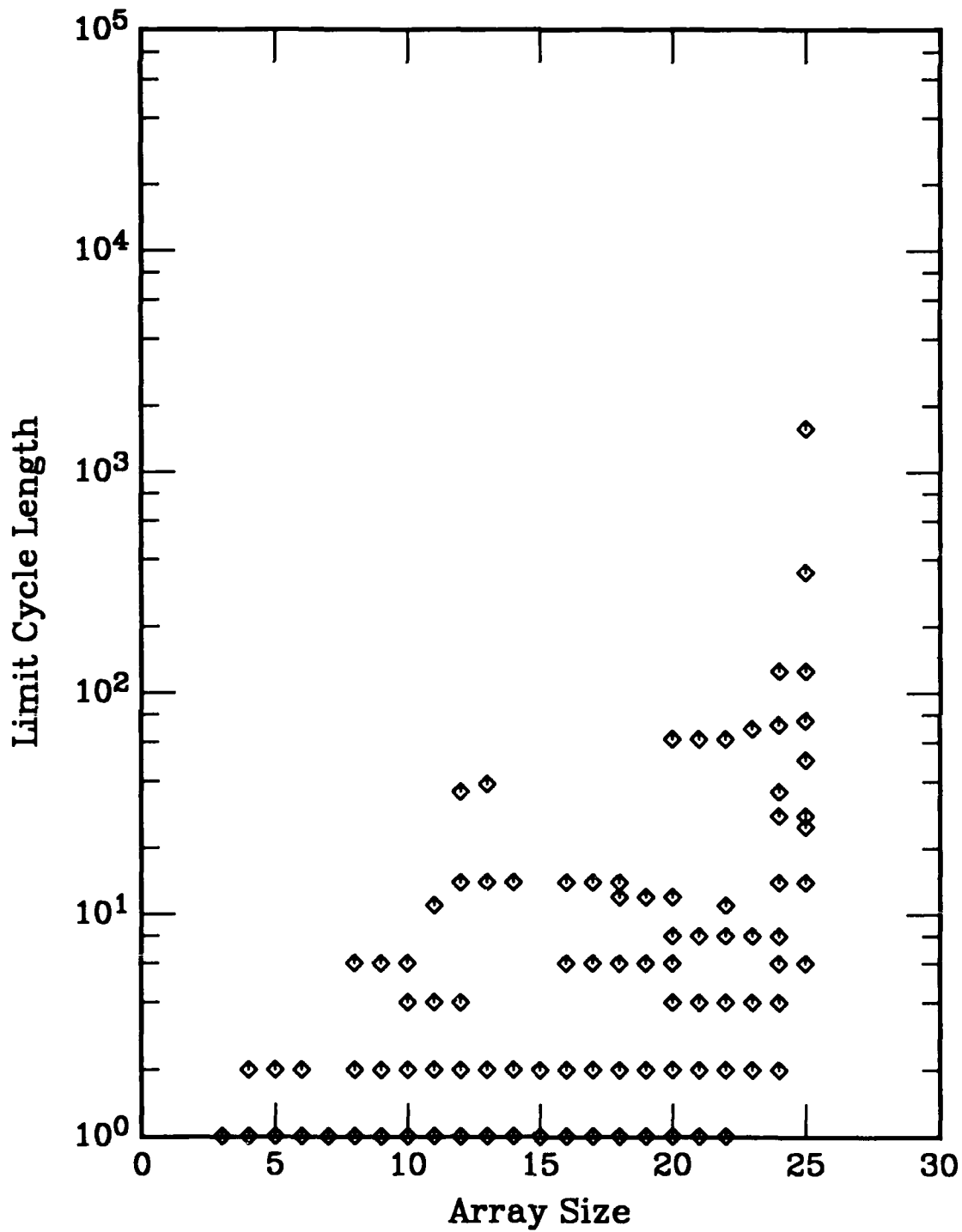


Figure C60. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 126

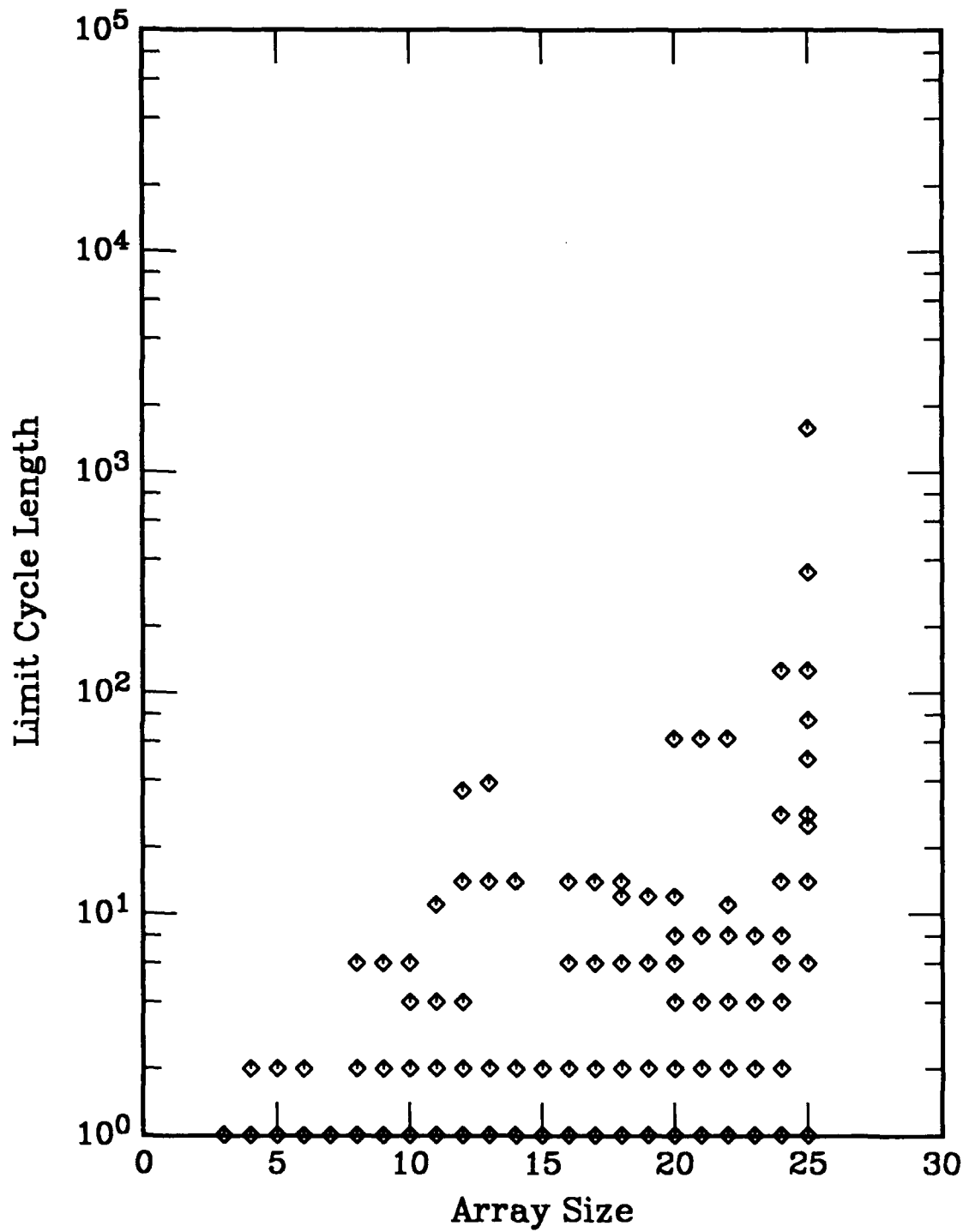


Figure C61. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 128

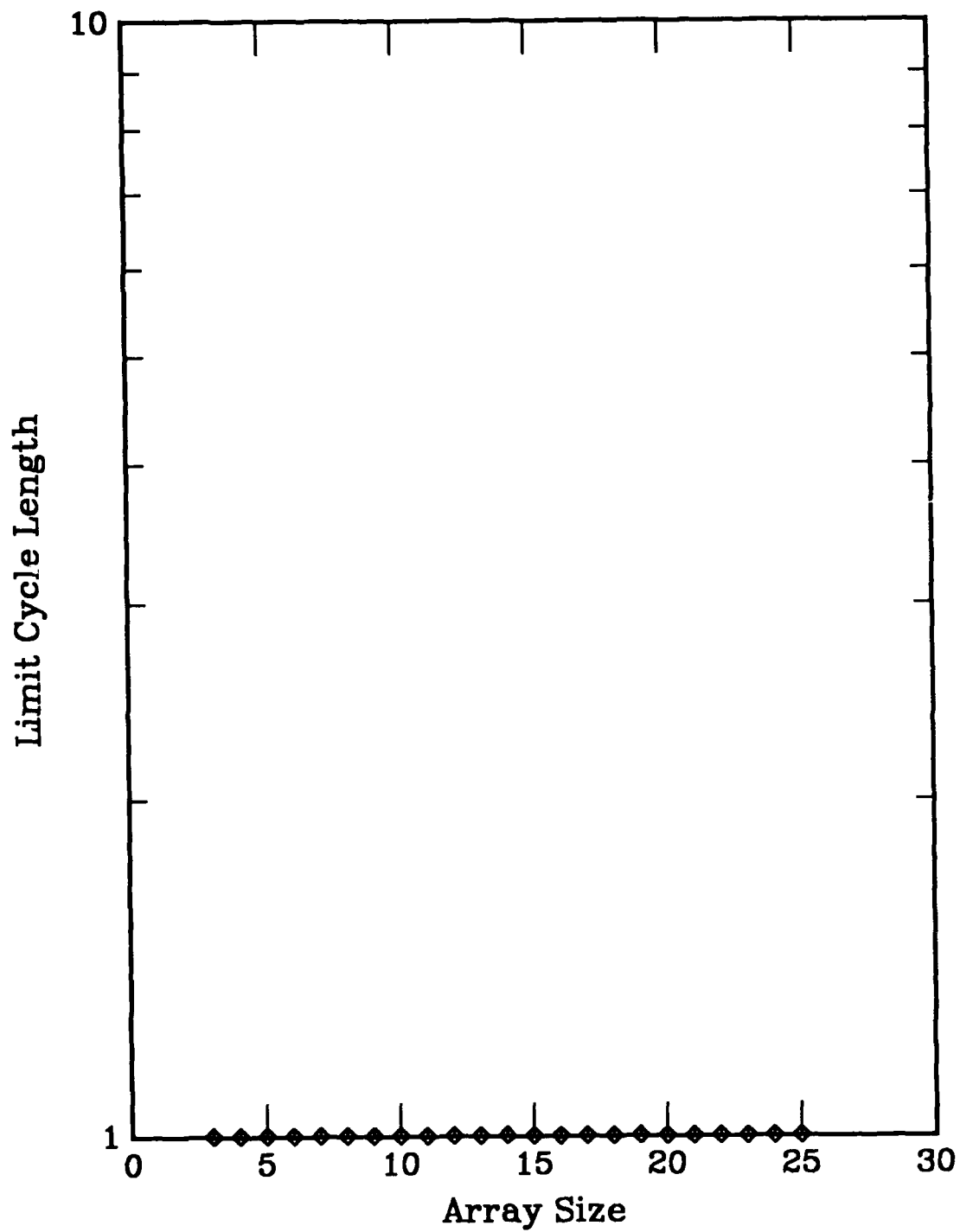


Figure C62. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 130

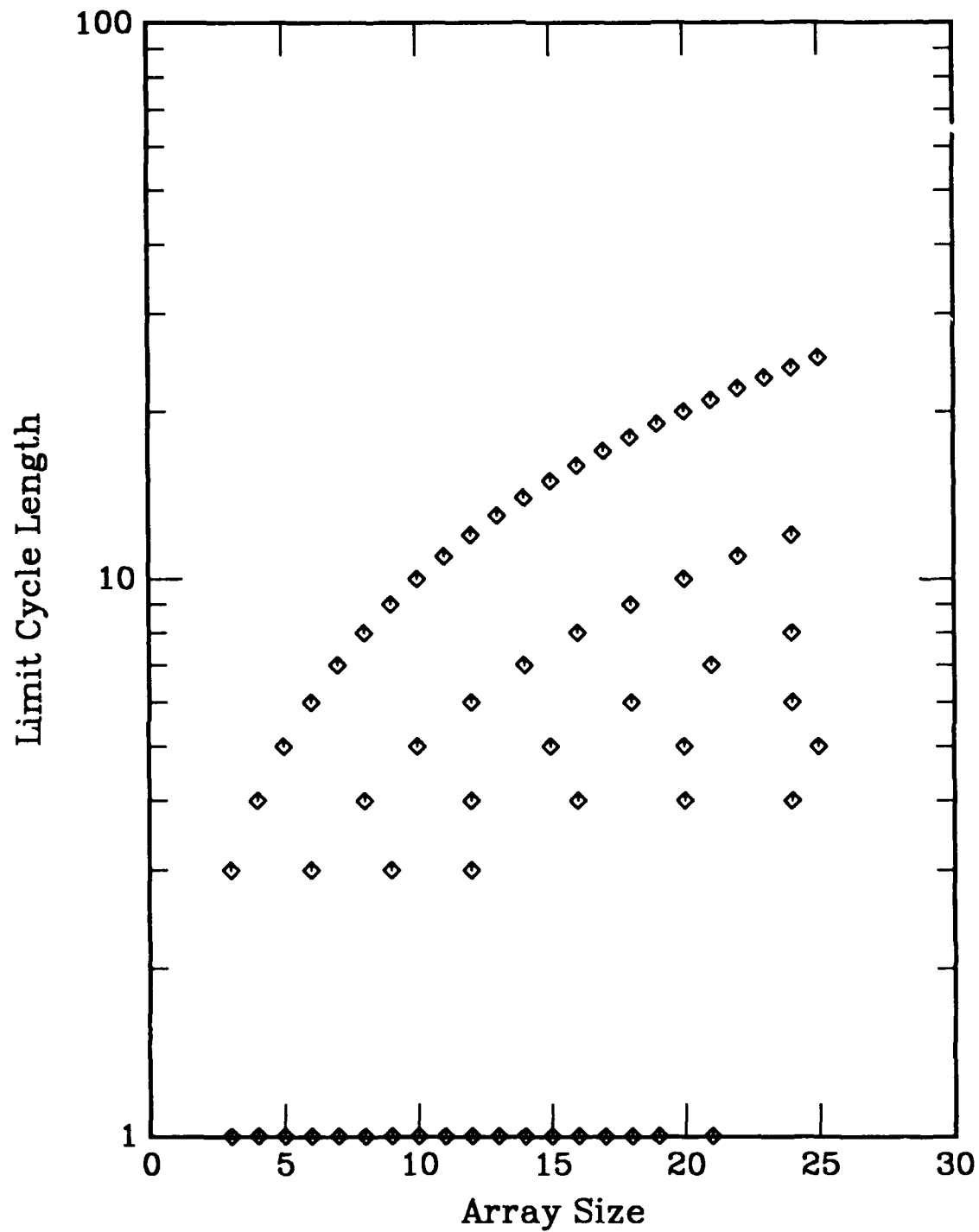


Figure C63. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 132

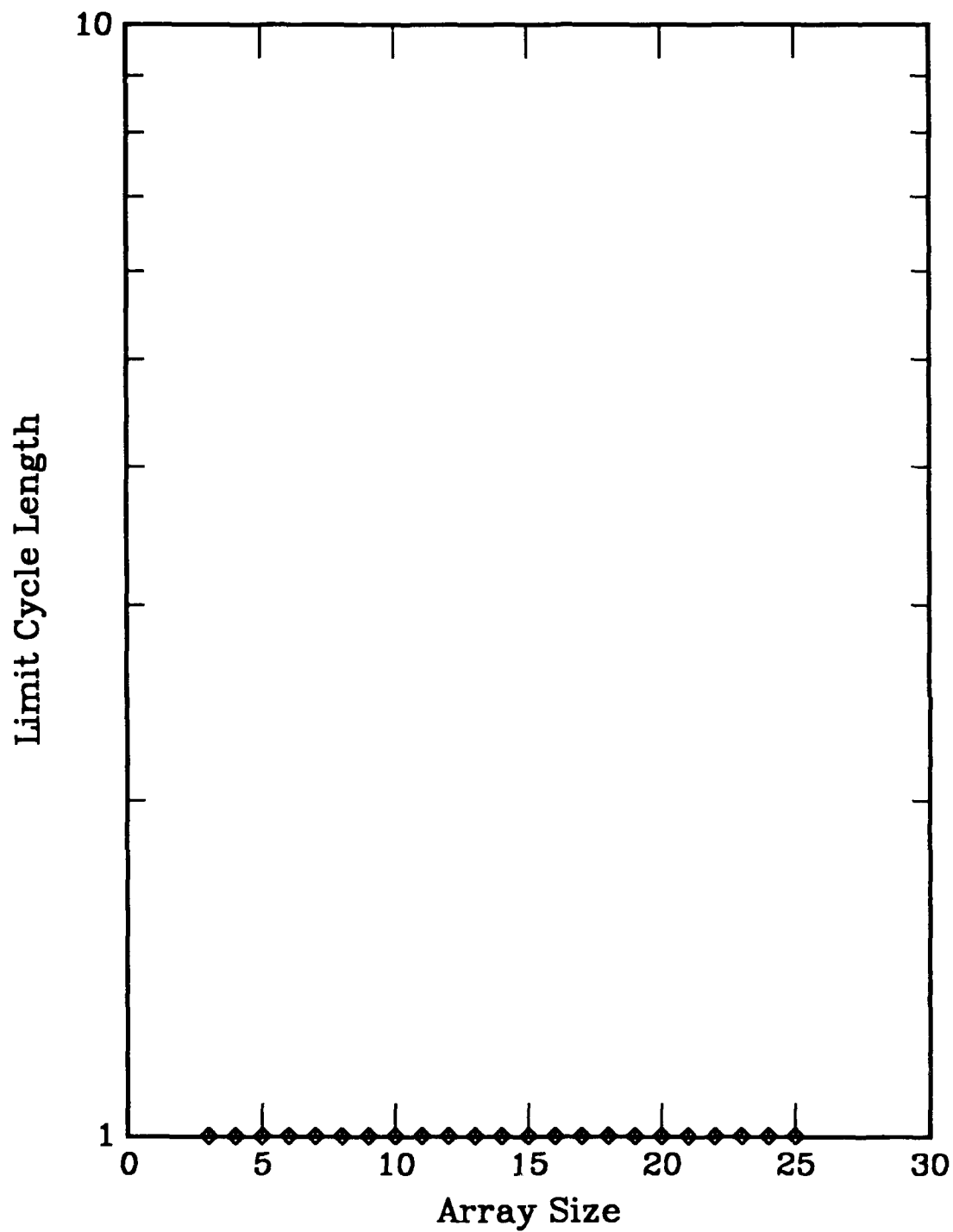


Figure C64. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

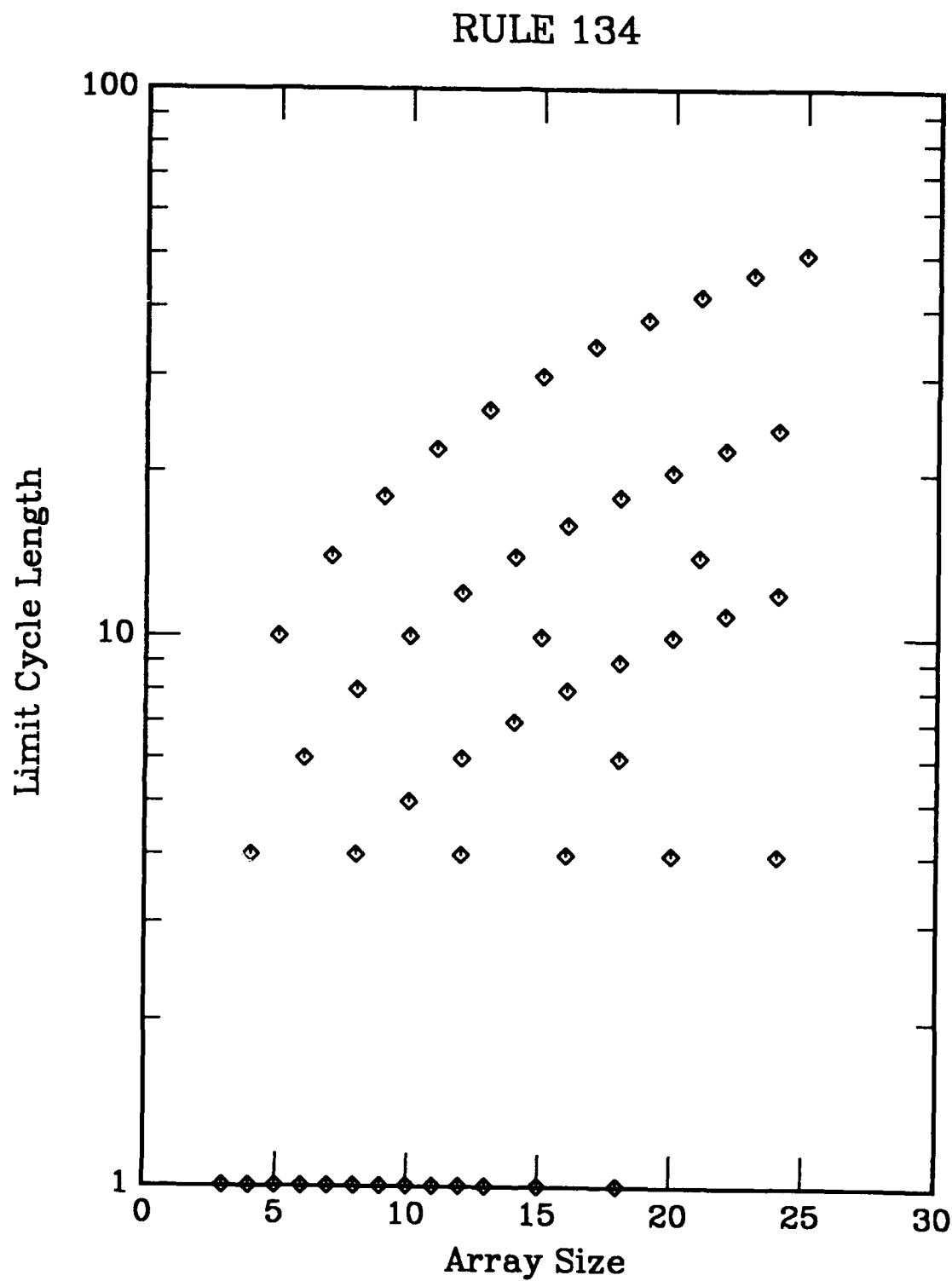


Figure C65. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 136

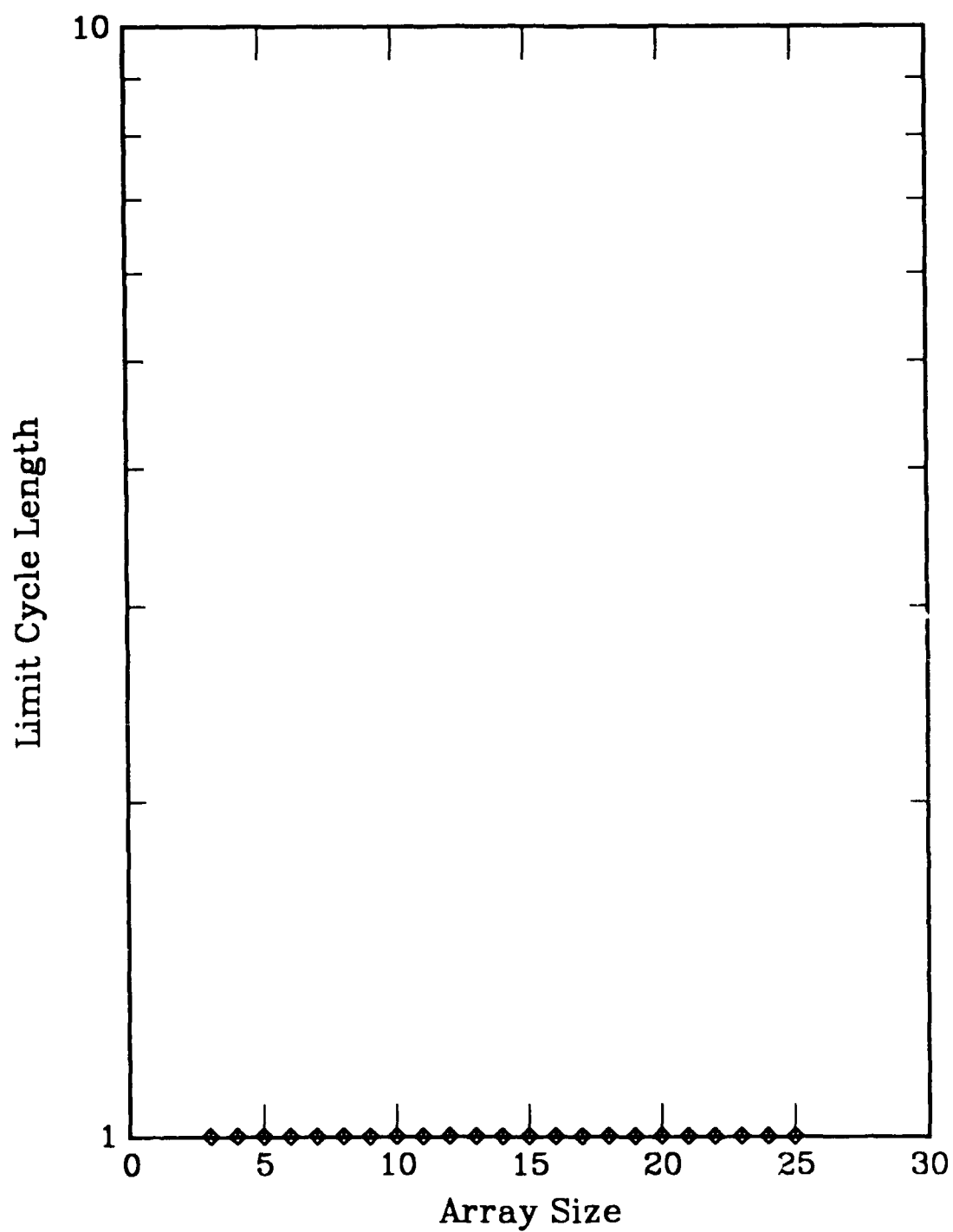


Figure C66. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 138

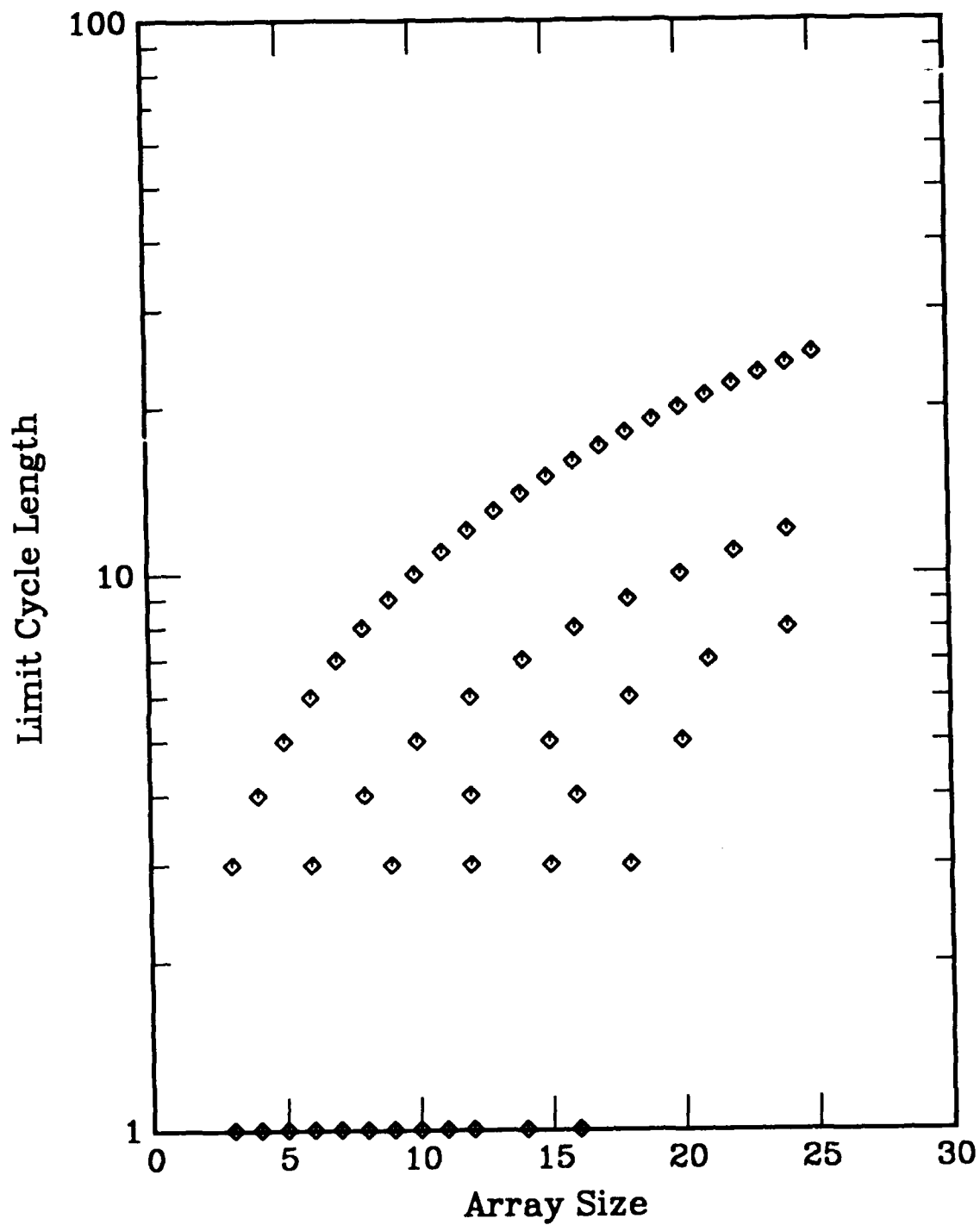


Figure C67. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 140

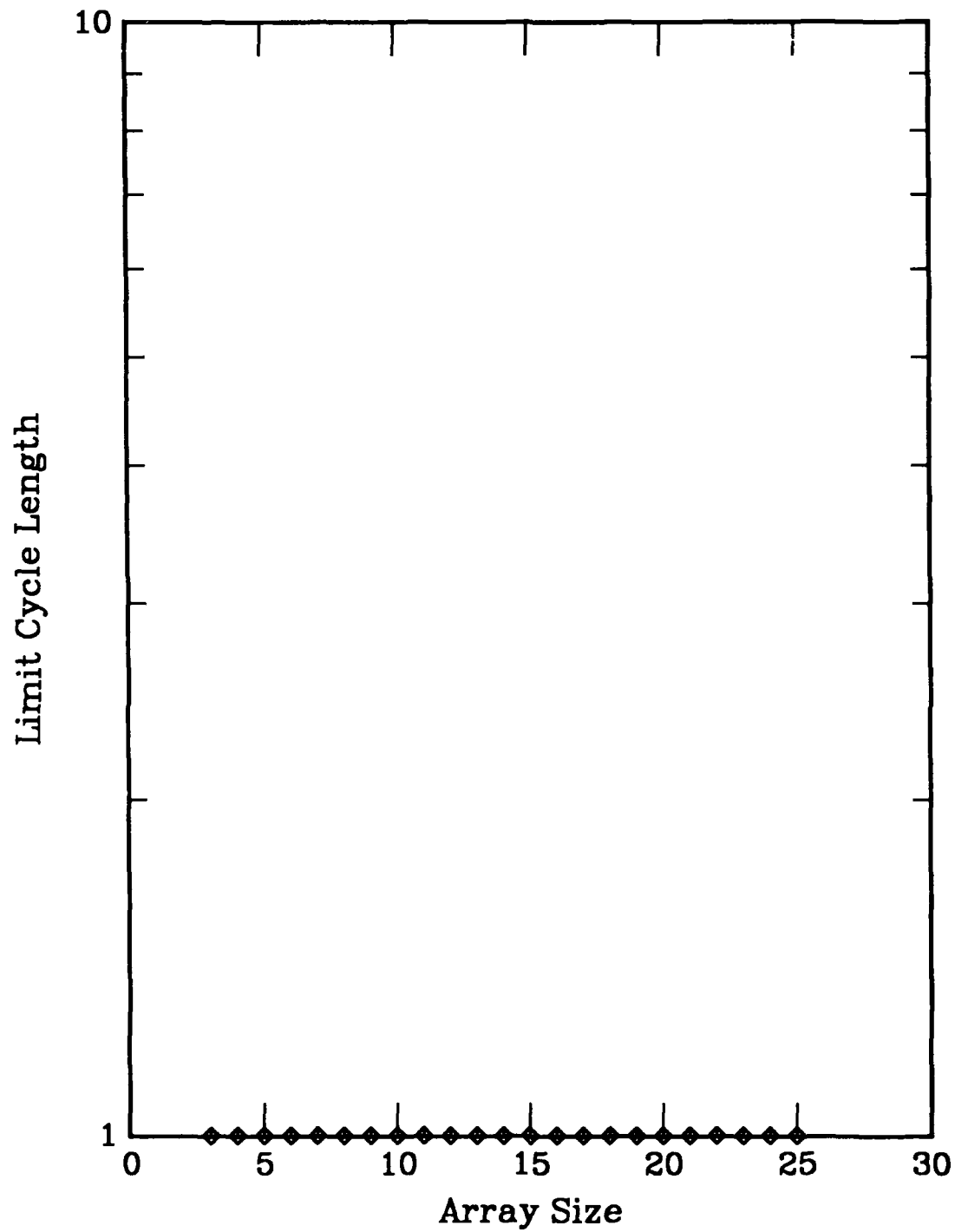


Figure C68. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 142

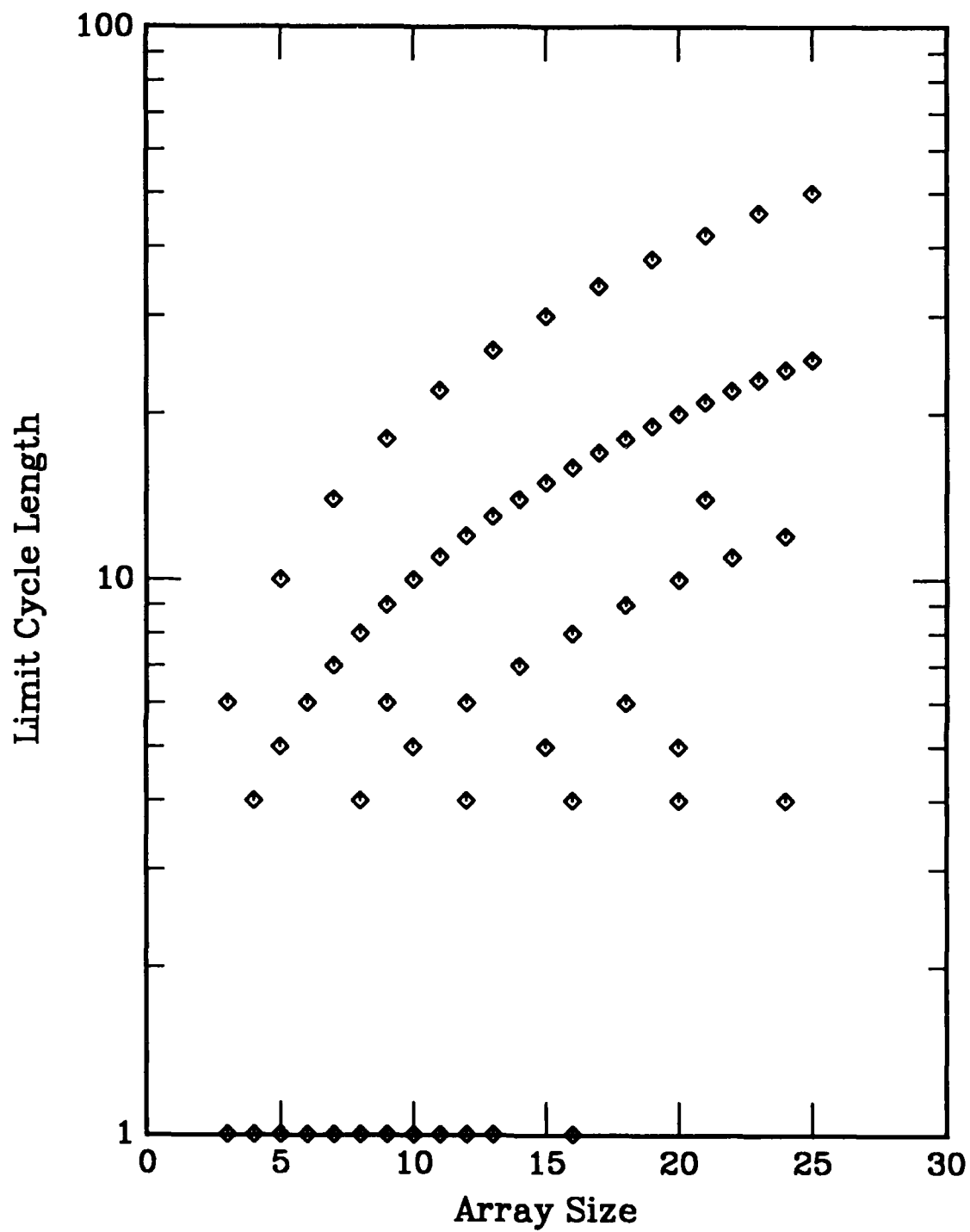


Figure C69. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 146

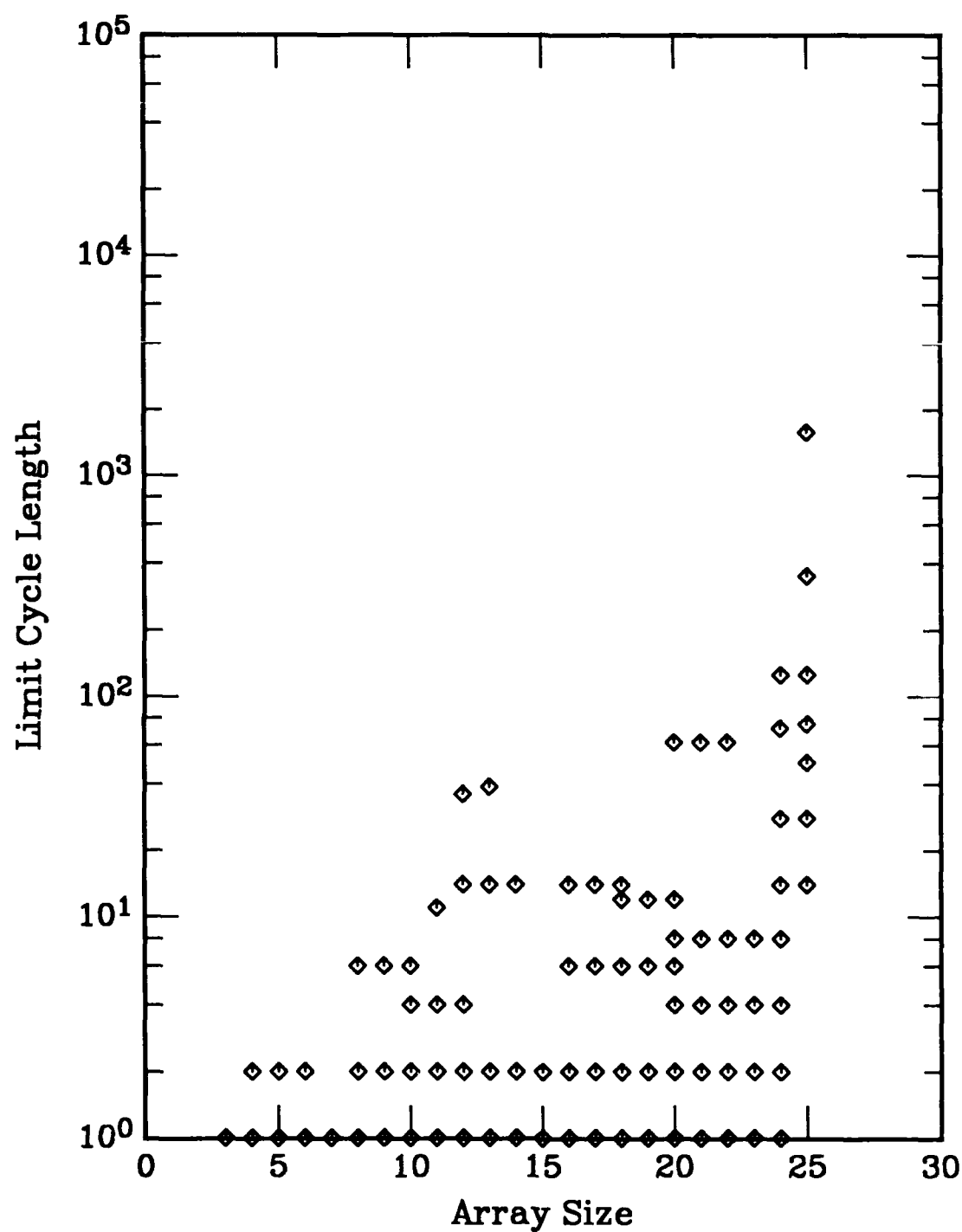


Figure C70. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 150

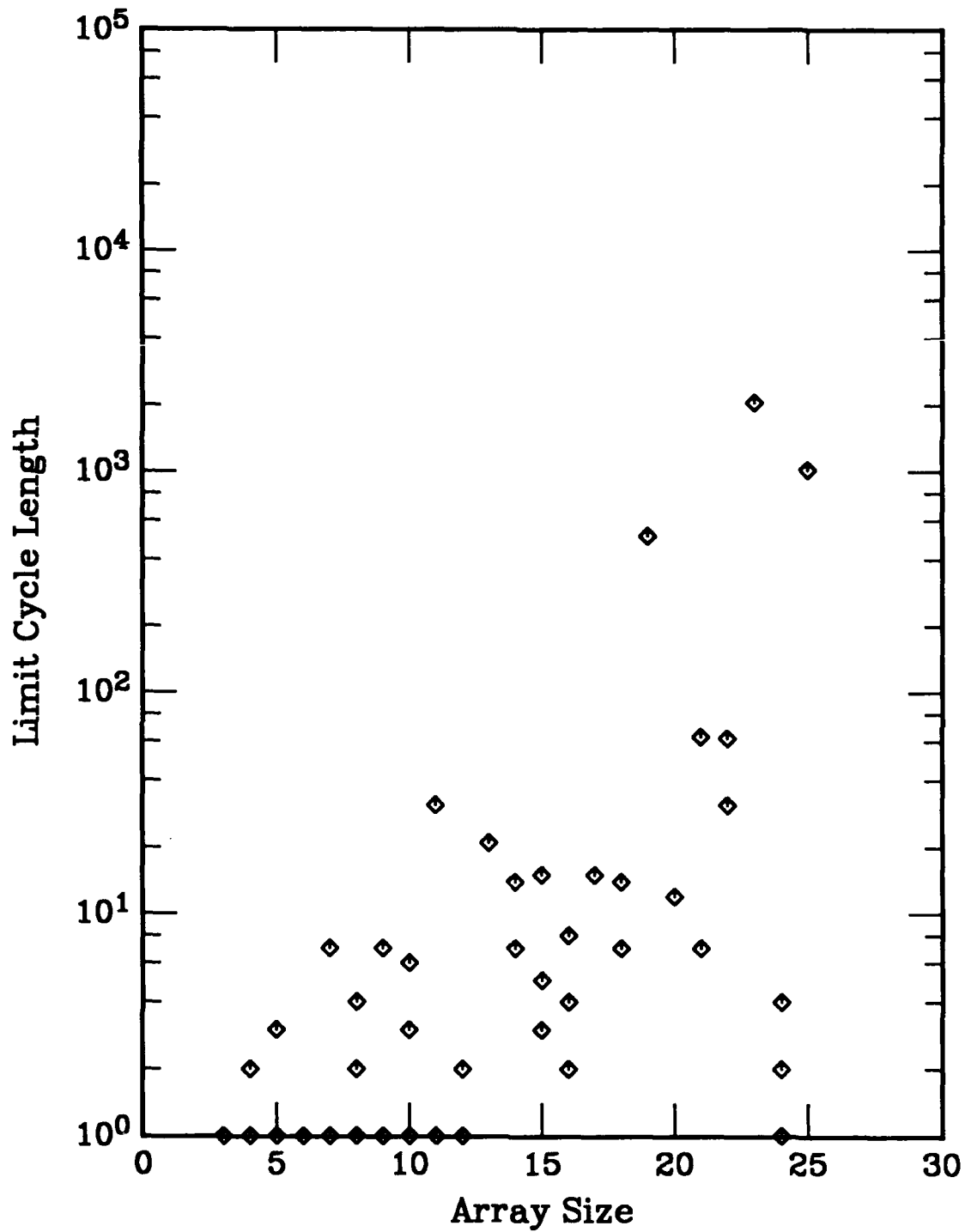


Figure C71. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 152

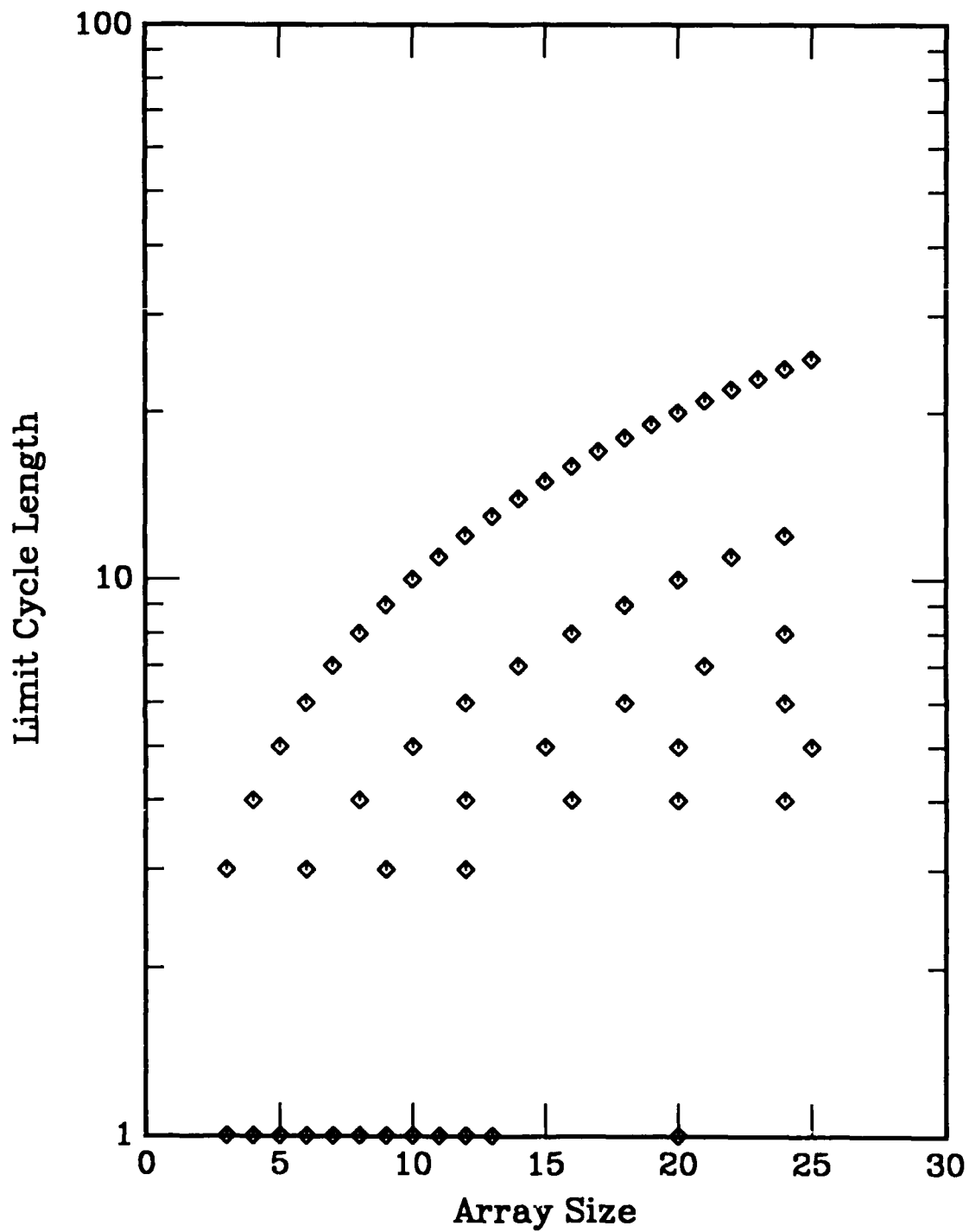


Figure C72. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

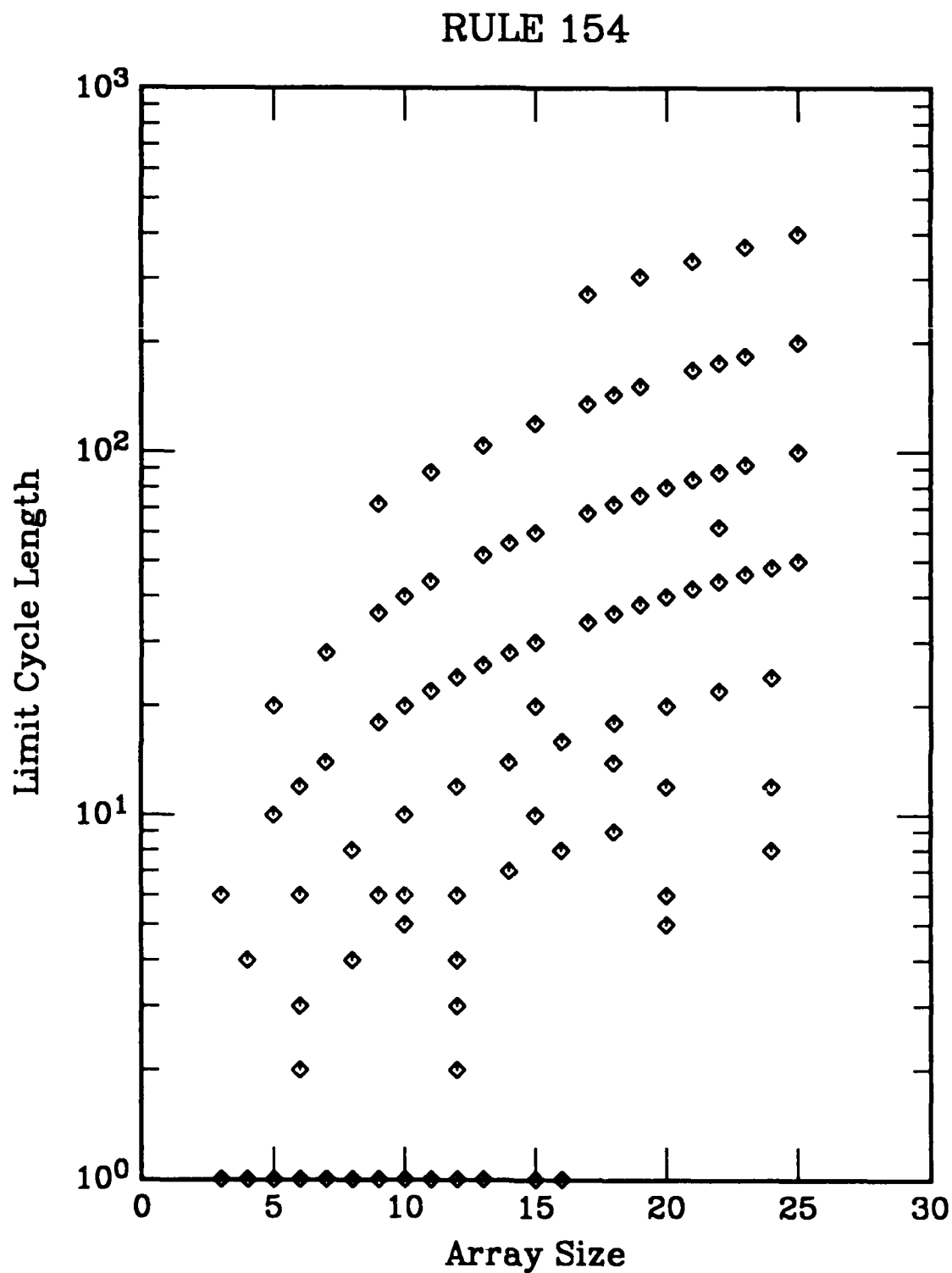


Figure C73. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 156

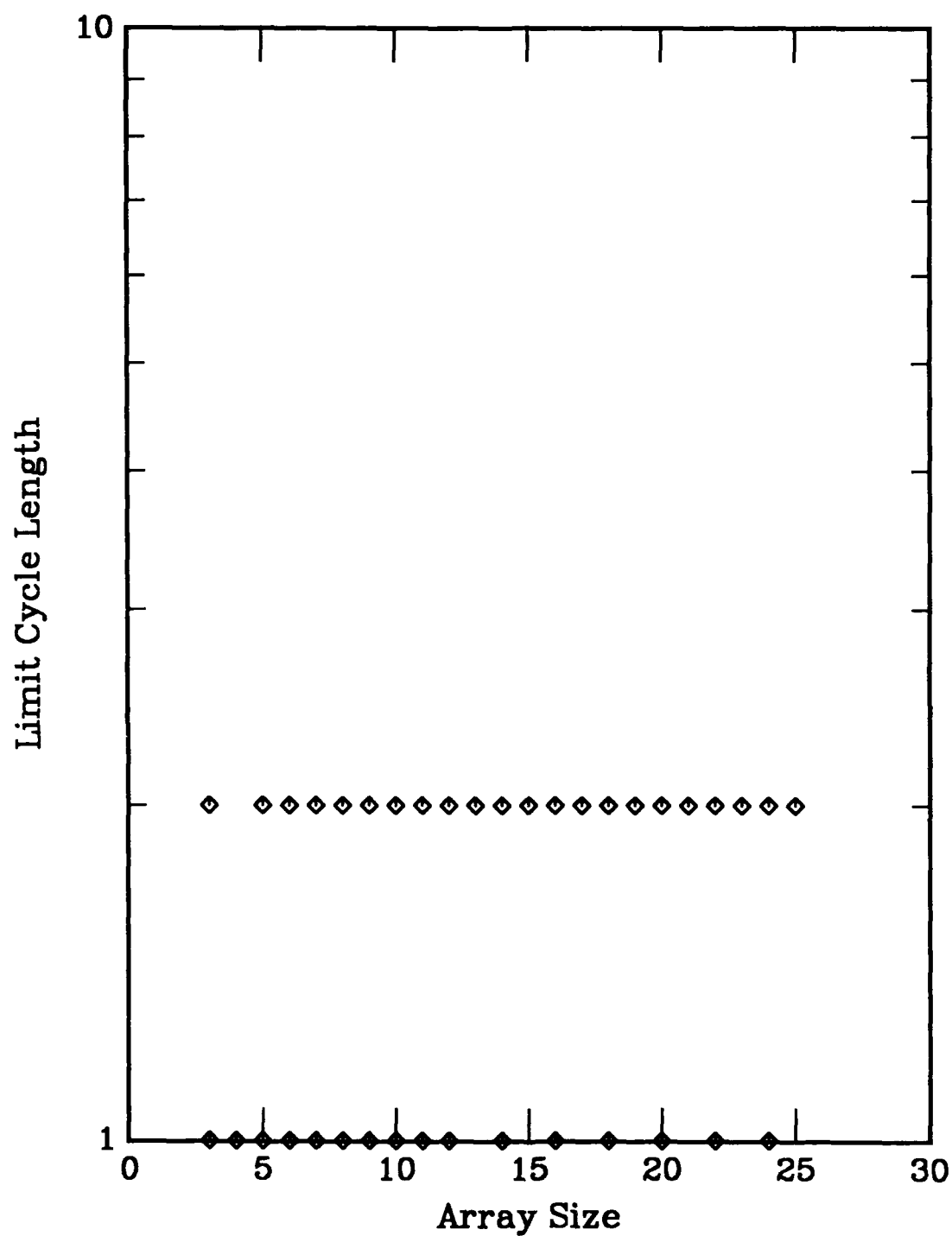


Figure C74. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 160

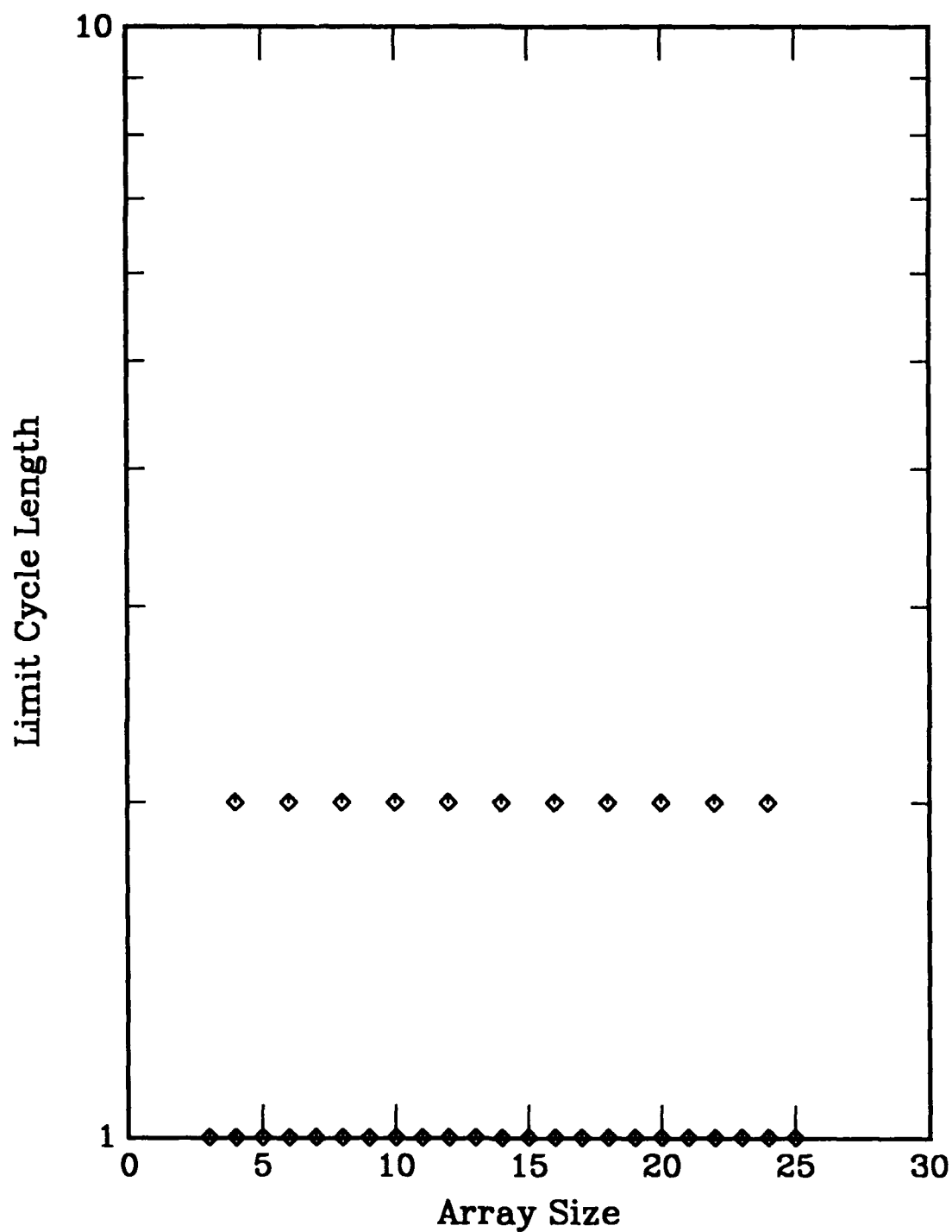


Figure C75. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 162

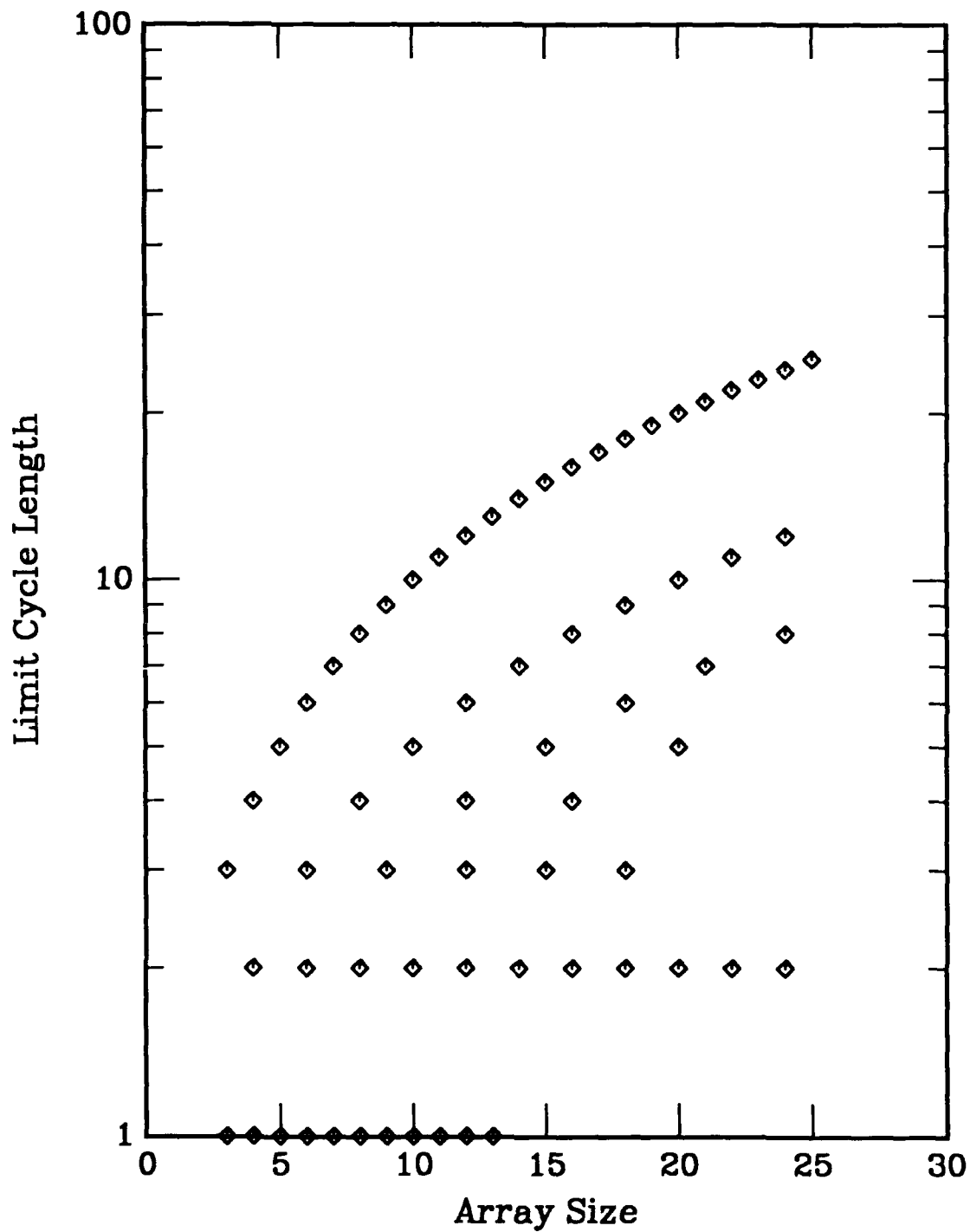


Figure C76. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 164

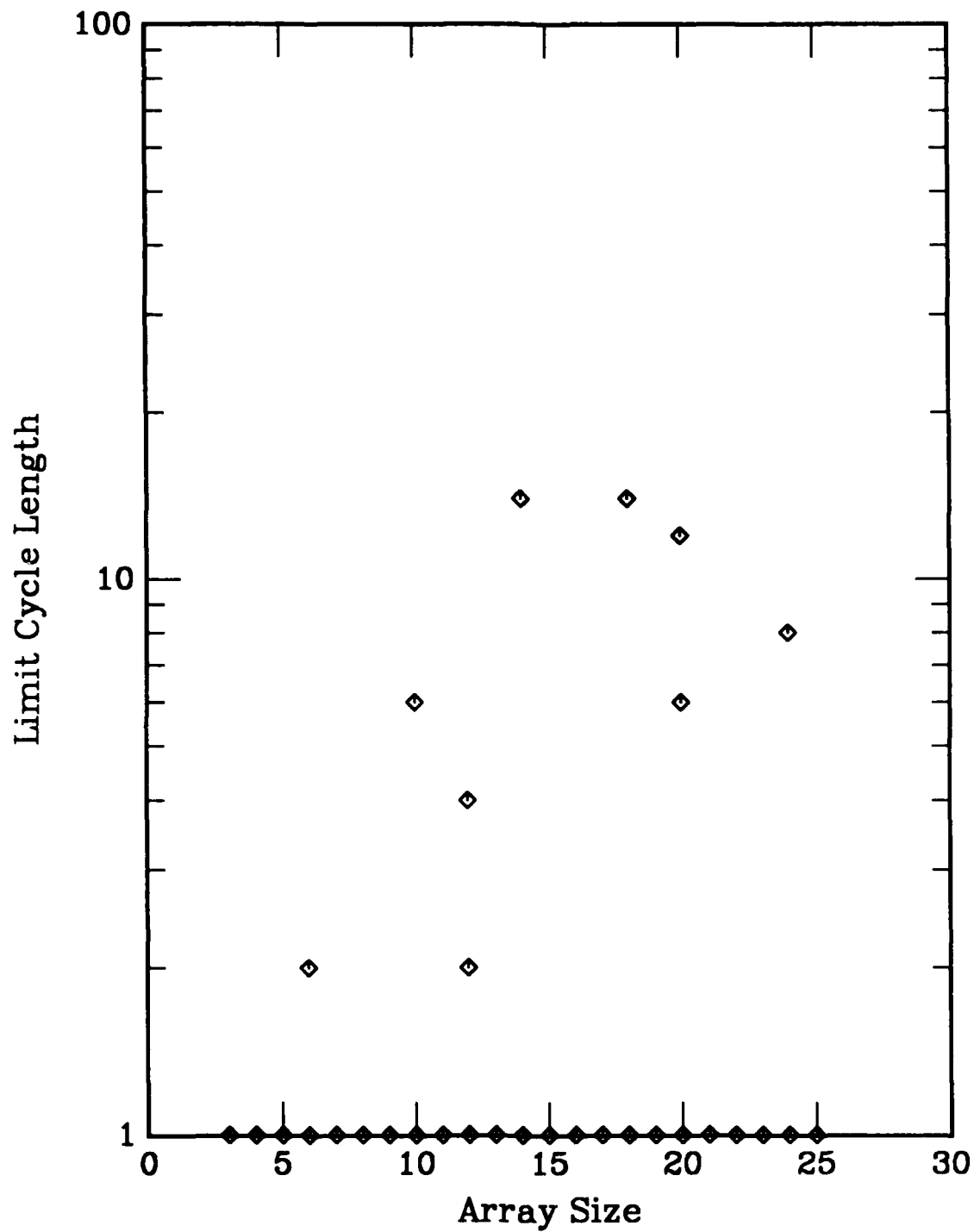


Figure C77. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 168

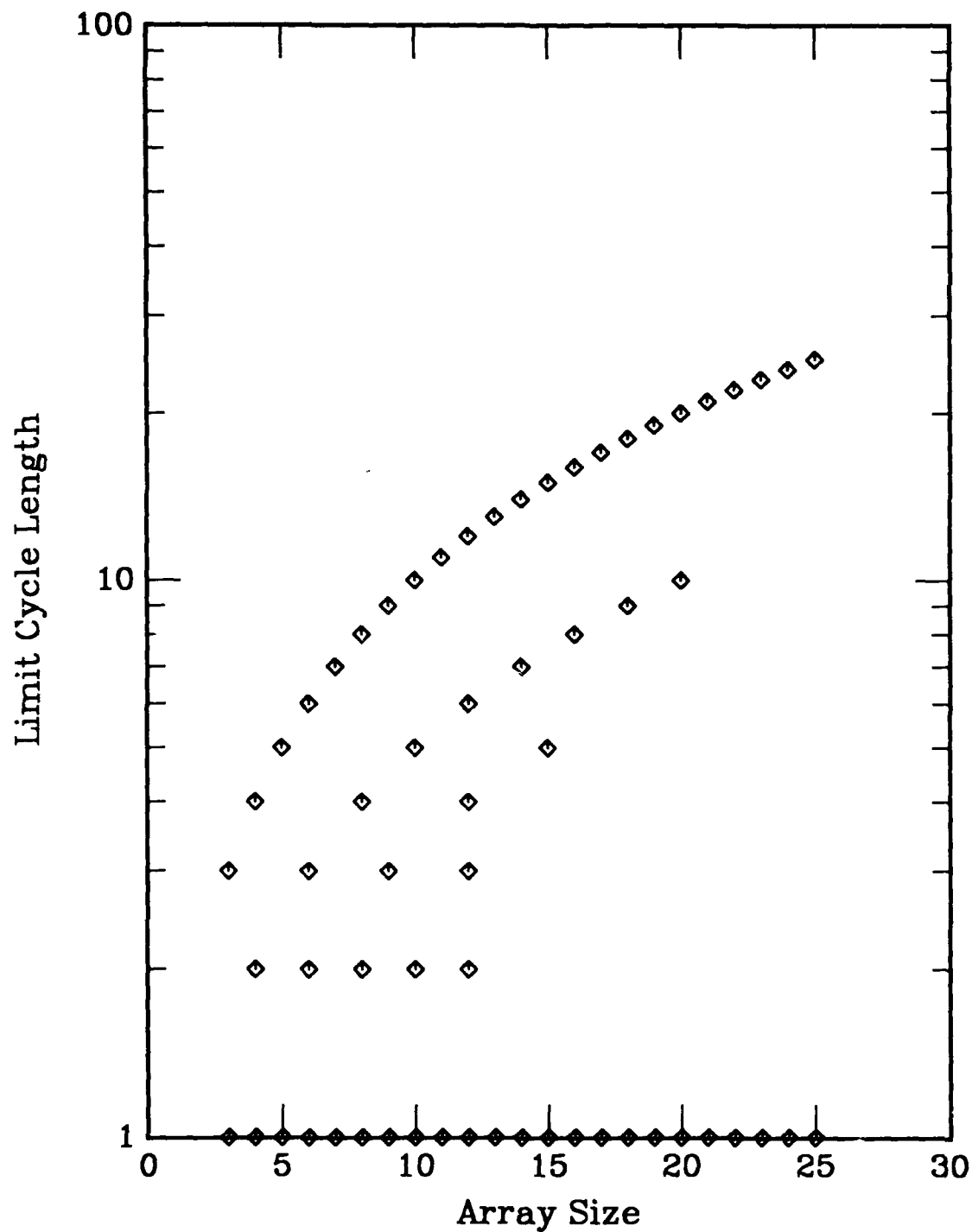


Figure C78. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 170

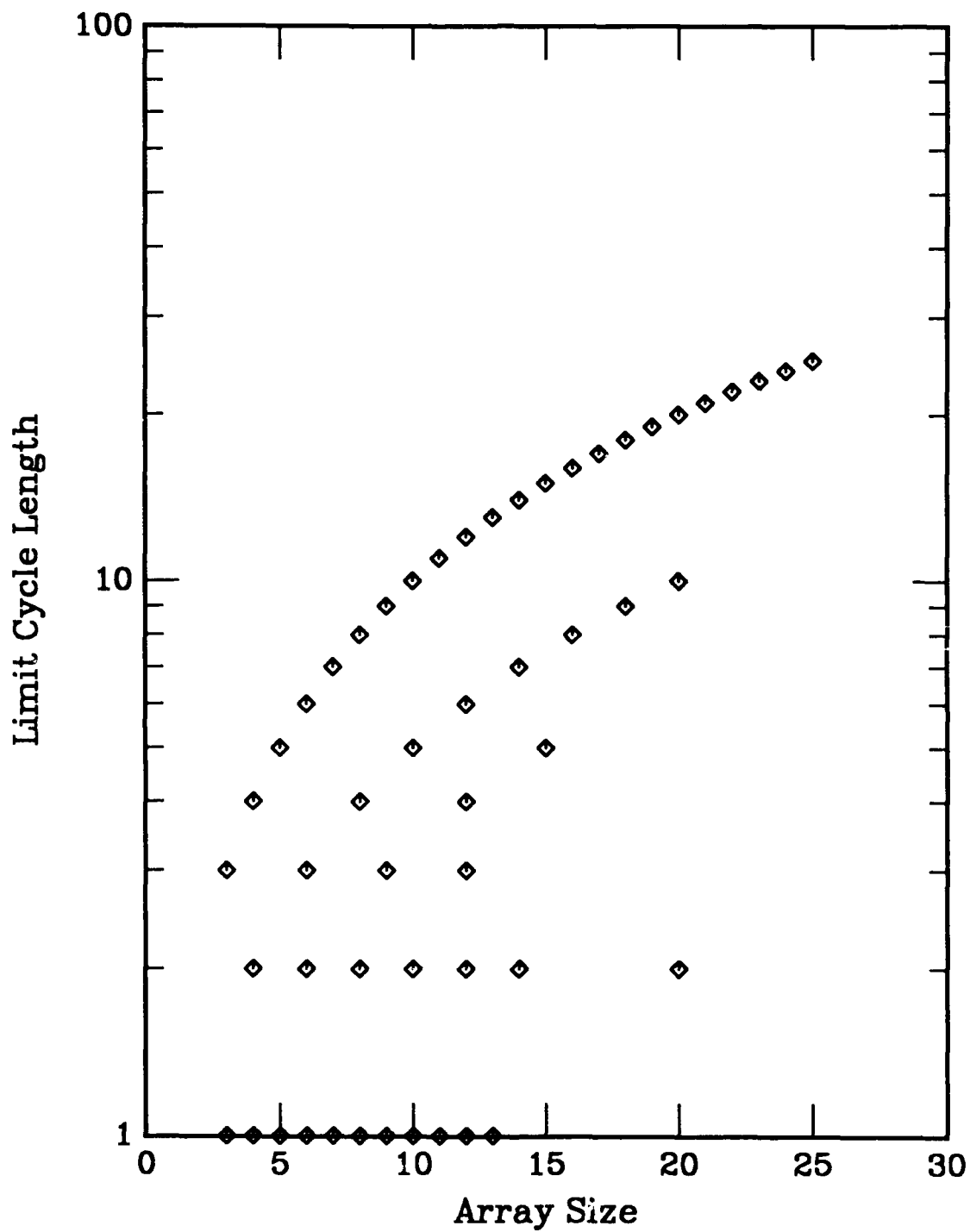


Figure C79. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 172

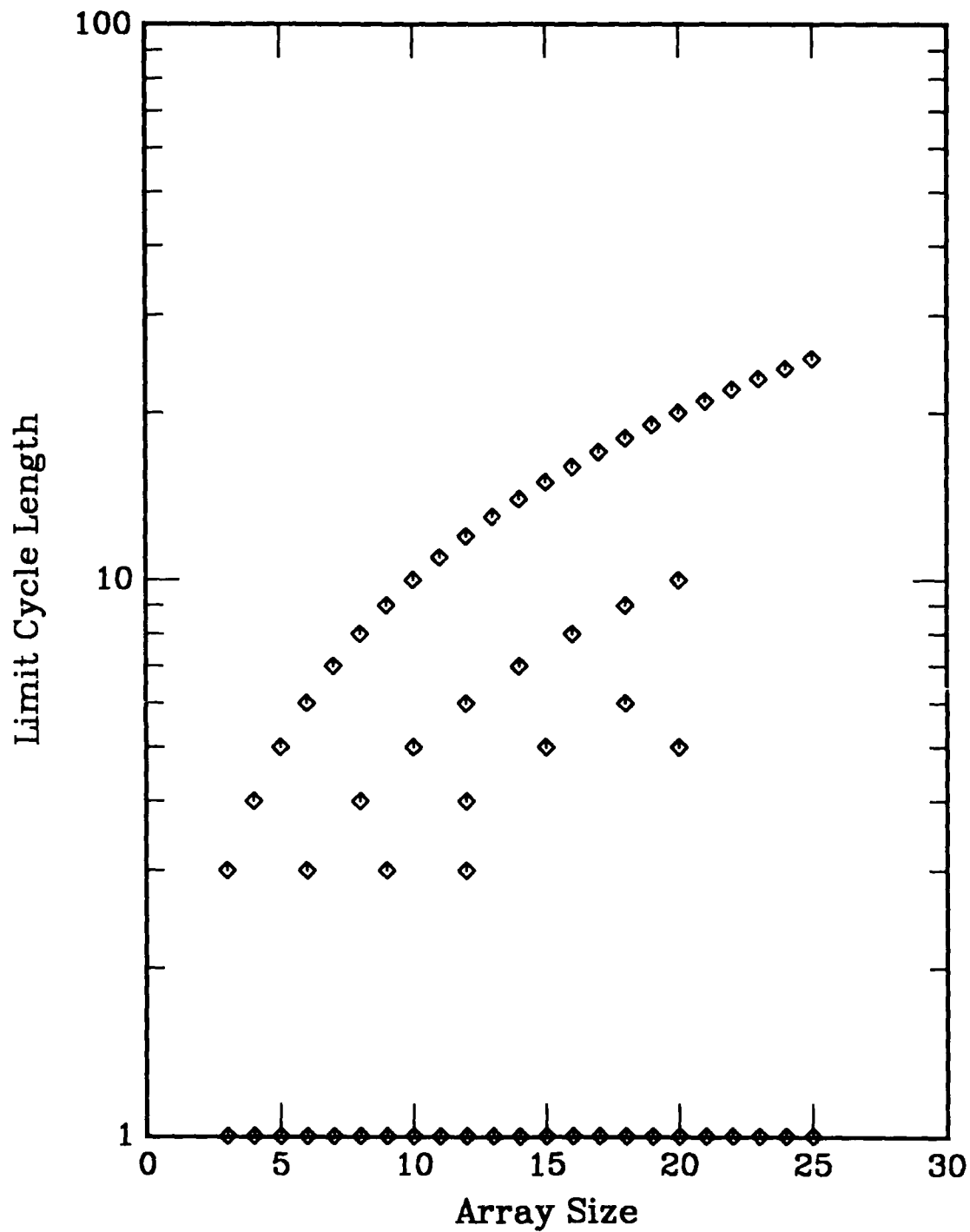


Figure C80. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 178

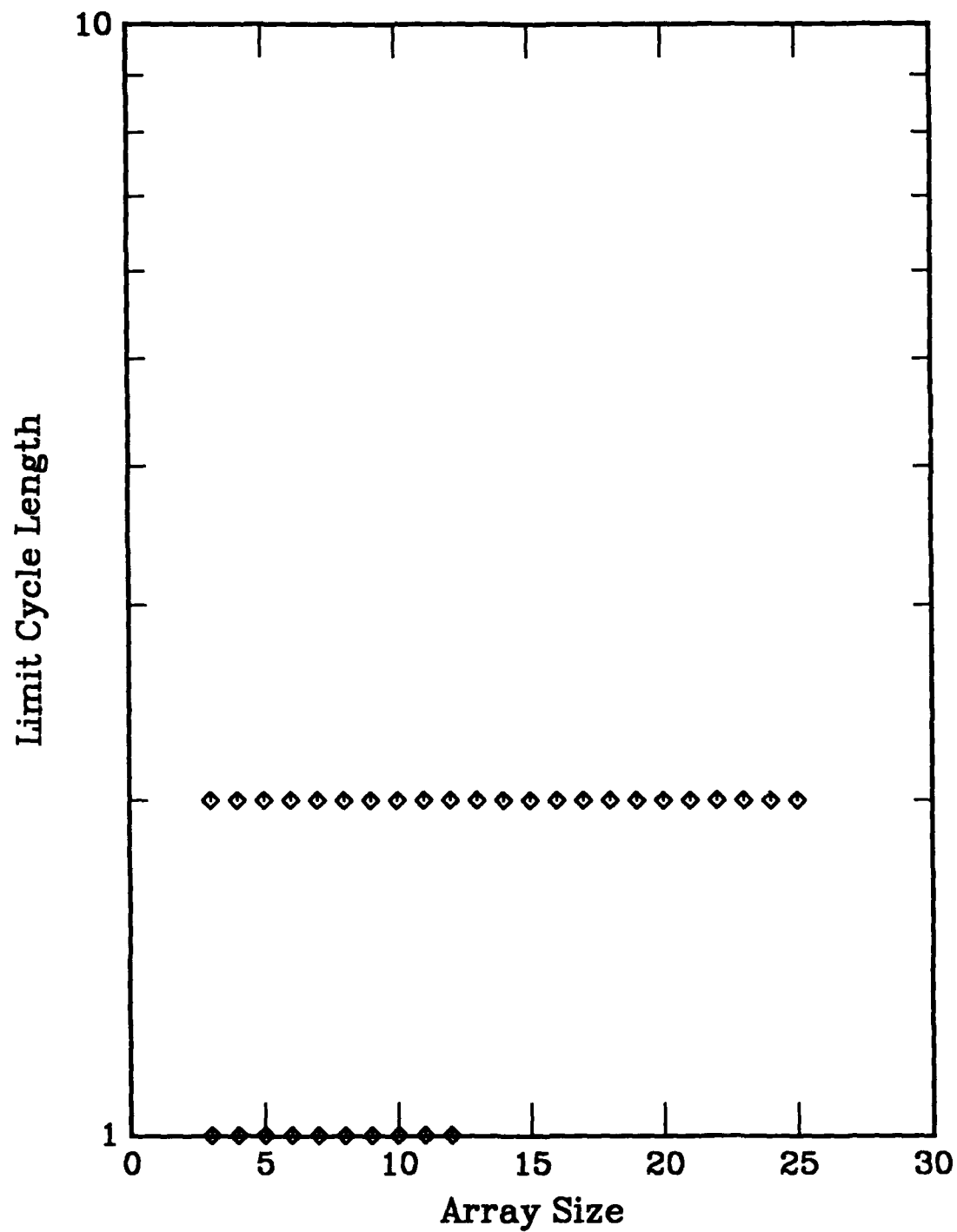


Figure C81. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 184

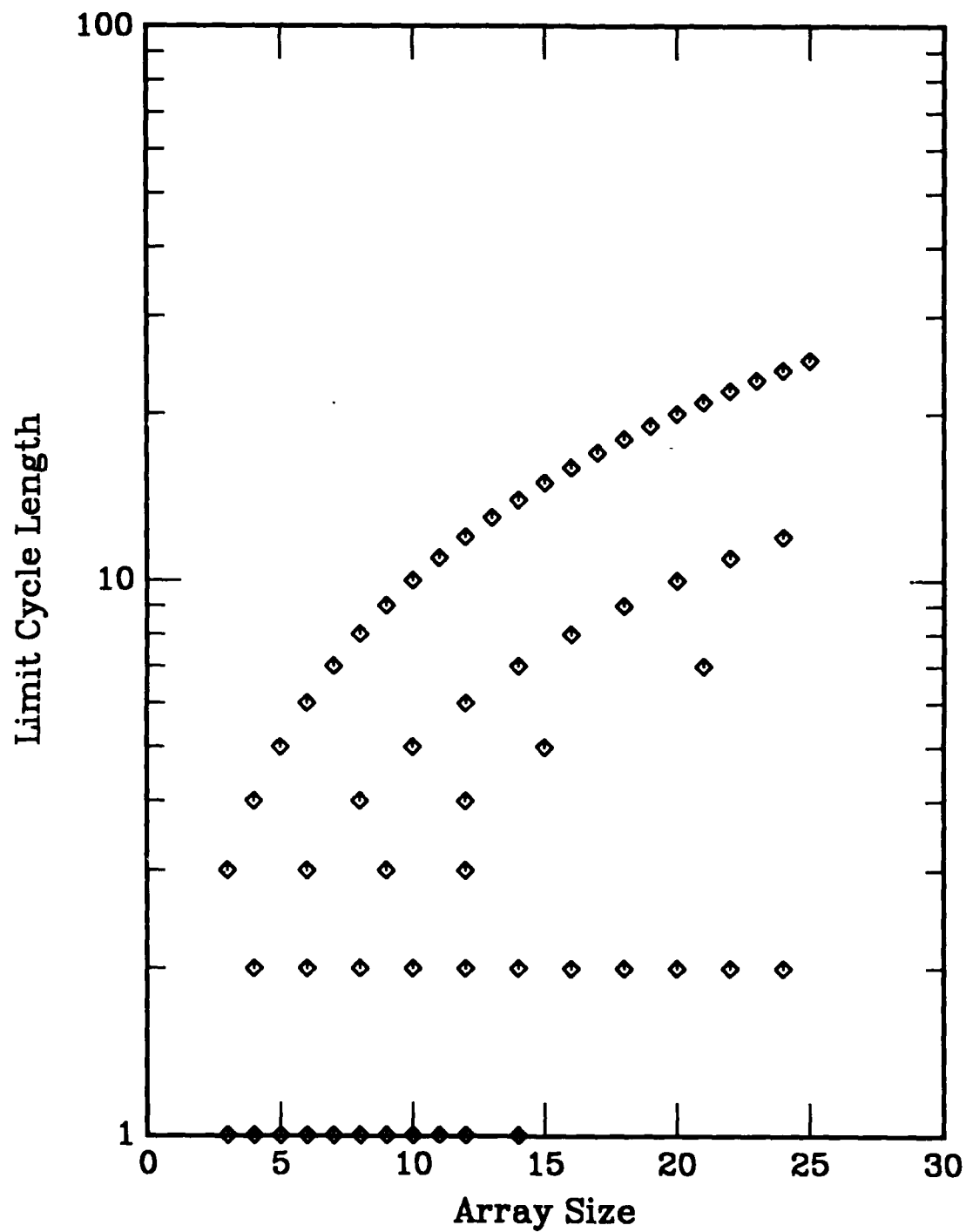


Figure C82. Limit cycle length vs array size for a one-dimensional, nearest neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 200

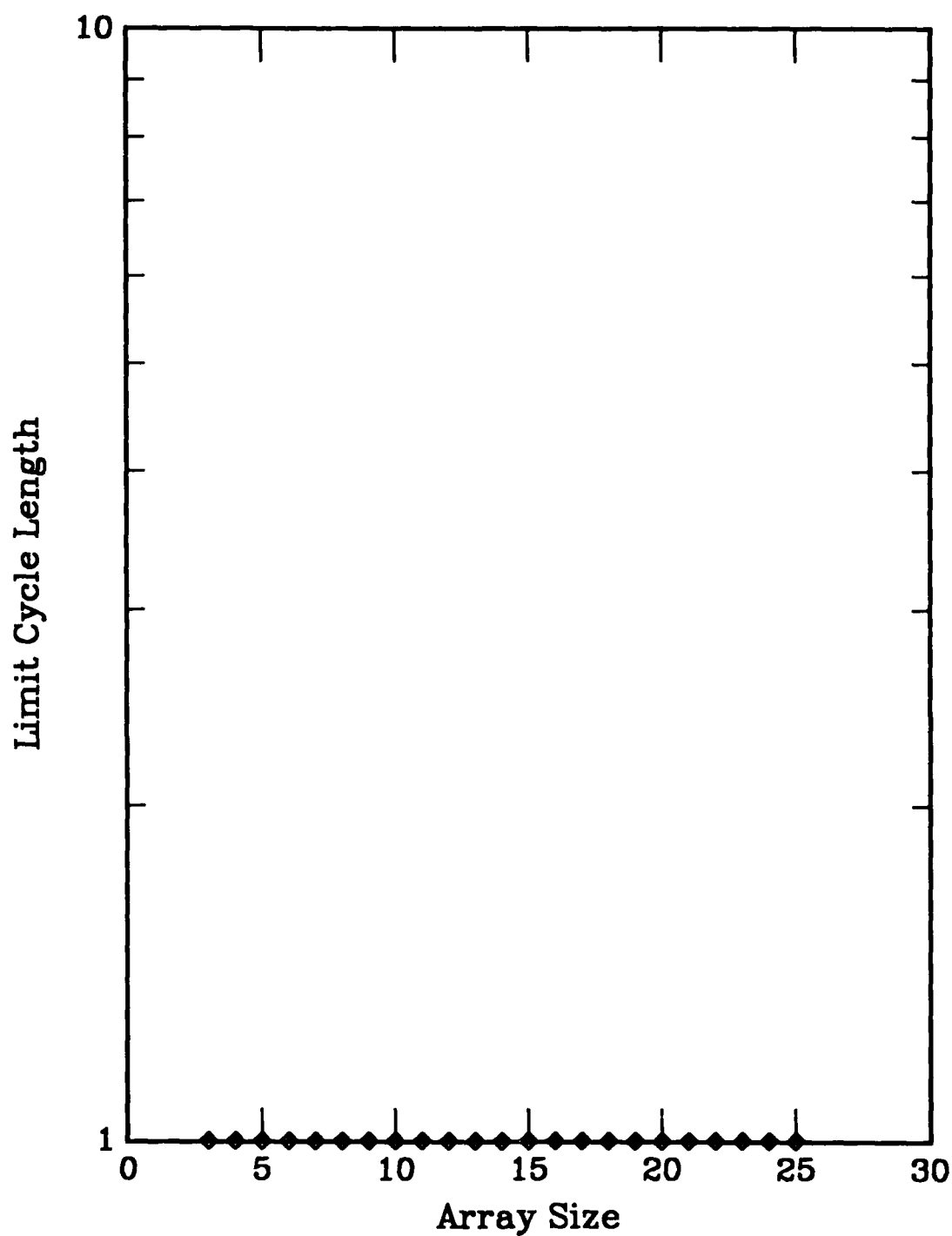


Figure C83. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 204

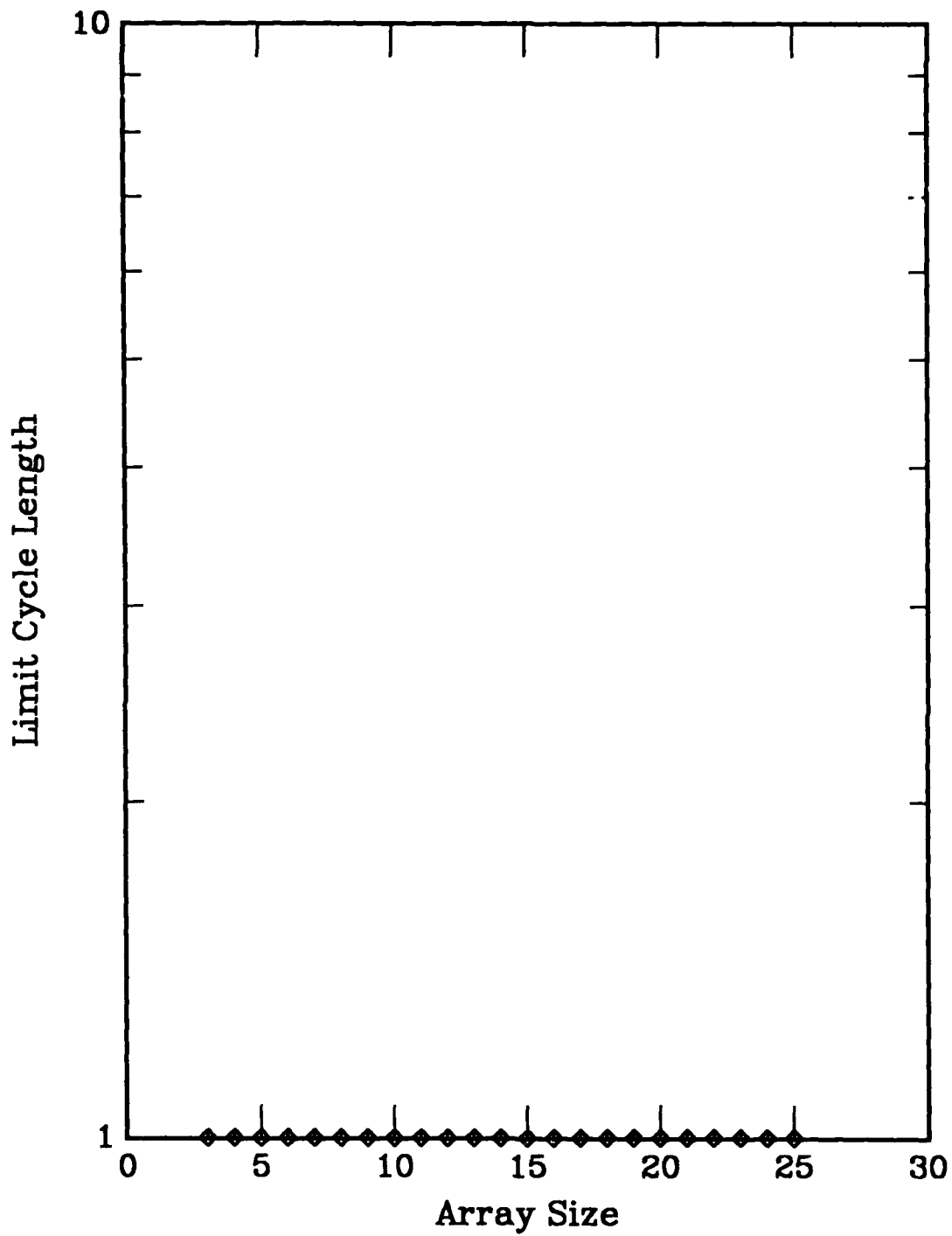


Figure C84. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

RULE 232

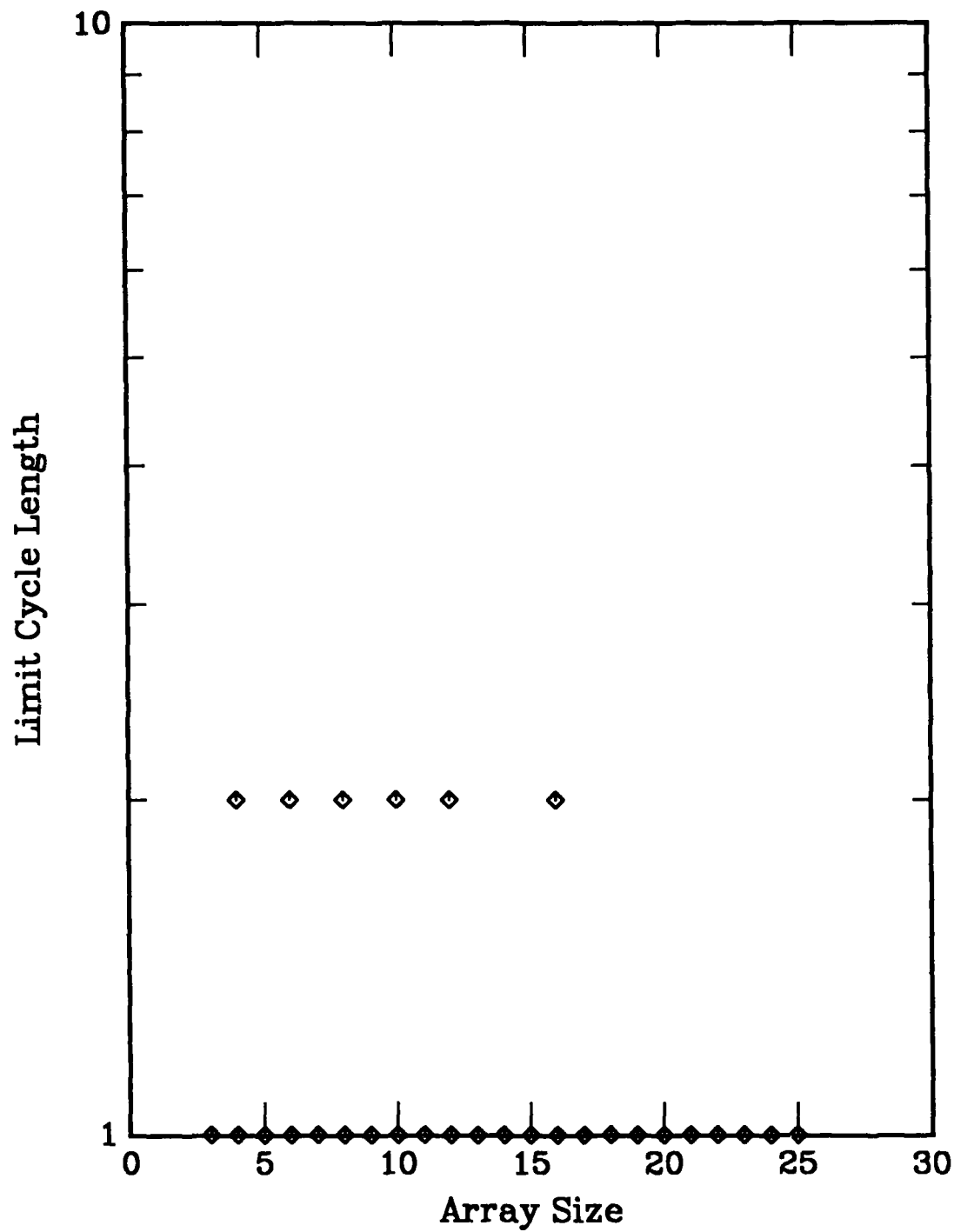


Figure C85. Limit cycle length vs array size for a one-dimensional, nearest-neighbor coupled cellular automata. Rule is defined in Appendix A.

APPENDIX D

ATTRACTOR BASIN VOLUMES FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

APPENDIX D

ATTRACTOR BASIN VOLUMES FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

This section catalogs the basin size of state attractors for nearest-neighbor coupled cellular automata. The probability (Z) that a random cell pattern will evolve to a given limit cycle is plotted versus limit cycle size (Y) and array size (X). This probability is equal to the fractional volume of state space absorbed by the attractor's basin. In simulating the model dynamics, the ends of the linear array of cells were coupled together so that the cells formed a closed loop. That is, the edge cells are actually nearest neighbors in the simulations. Of the 256 possible nearest-neighbor cell rules, only 88 rule operations are independent. Since for example, rules 14, 84, 143, and 213 lead to identical attractor spectra, only rule 14 is included in the plots. In addition, several independent rules lead to very uninteresting attractor probability spectra. In particular, we did not include any rule which was characterized by a single attractor. Rule 204 is an example of an independent rule which merely replicated the starting cell pattern over time. The attractor spectra of such rules are trivial to predict. Refer to the Appendices A and B for a complete list of independent rules and their attractor sets.

The limiting dynamics falls into four simple classes of behavior, distinguishable by the various attractor types observed.

- Class J Rules (Type A1 attractors) : 4, 5, 8, 12, 13, 19, 23, 28, 29, 32, 33, 36, 44, 50, 51, 72, 76, 77, 78, 104, 128
- Class K Rules (Types A1 and A2 attractors) : 3, 6, 7, 9, 10, 11, 14, 15, 24, 27, 40, 42, 43, 46, 56, 57, 58, 62, 74, 130, 134, 138, 142, 152, 162, 184
- Class L Rules (Types A1 and A3 attractors) : 8, 30, 37, 45, 54, 60, 73, 90, 94, 105, 110, 122, 126, 146, 150, 164, 168, 170, 172
- Class M Rules (Types A1, A2, and A3 attractors) : 25, 26, 35, 38, 41, 106

where we have defined the attractor types as follows:

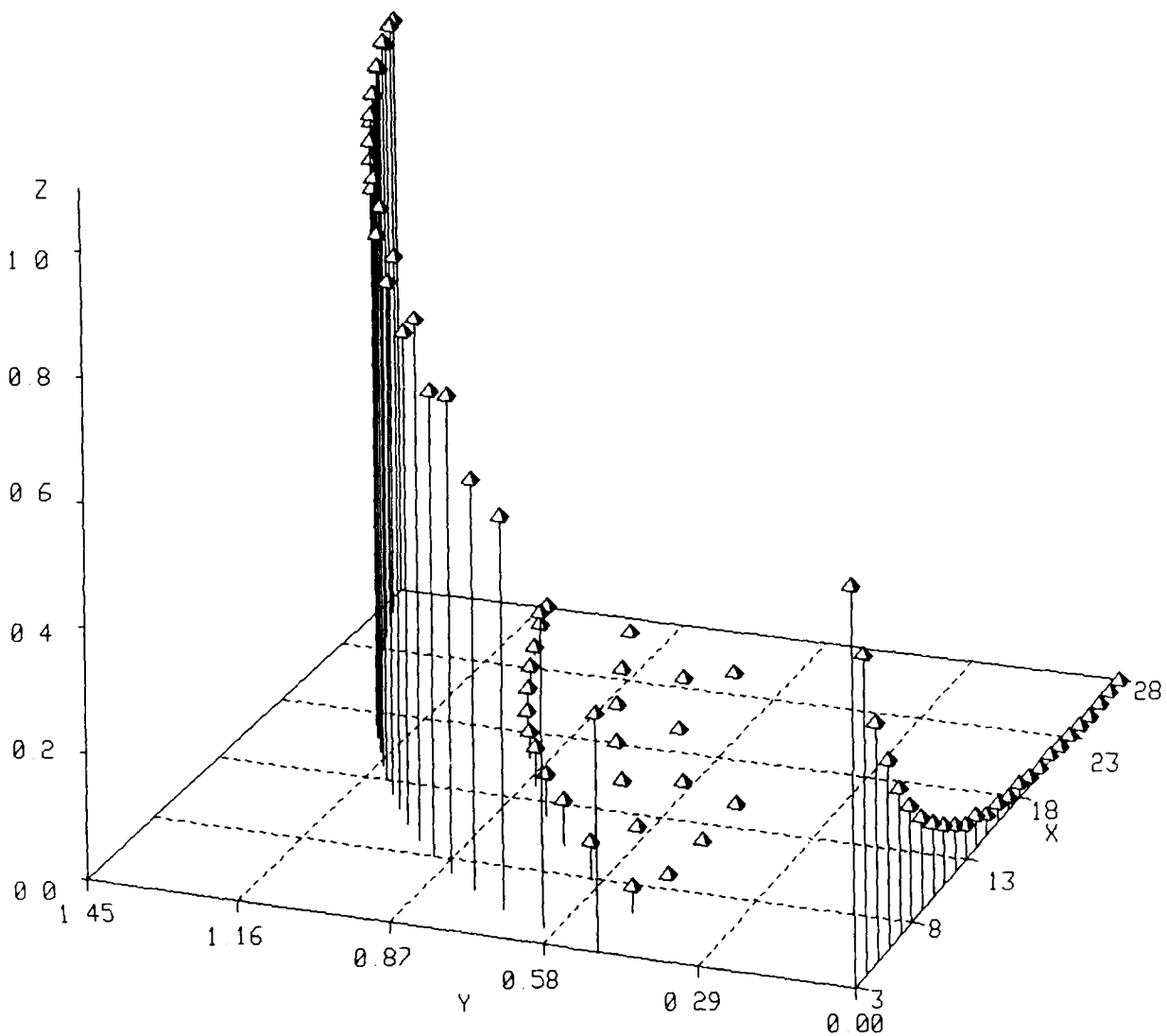
Type A1 Attractor : Cycle length is independent of array size.

Type A2 Attractor : Cycle length is in integral proportion to array size.

Type A3 Attractor : Cycle length is uncorrelated with array size.

Within the Class L and M rules is a subset of rules that contain attractors which approach full ergodicity. That is, a few limit cycle lengths are within an order of magnitude of the total volume of state space for the model. These rules are 30, 45, 54, 106, and 110. Rule 30 and 45 display attractors which are nearly perfect pseudo-random sequence generators. As shown in the respective plots, these attractors lie along the diagonal of each plot. The high attractor probabilities for these rules are indicative of the fact that this class of attractor absorbs practically all of the available state space for automata of these sizes and rule types.

Rule 2

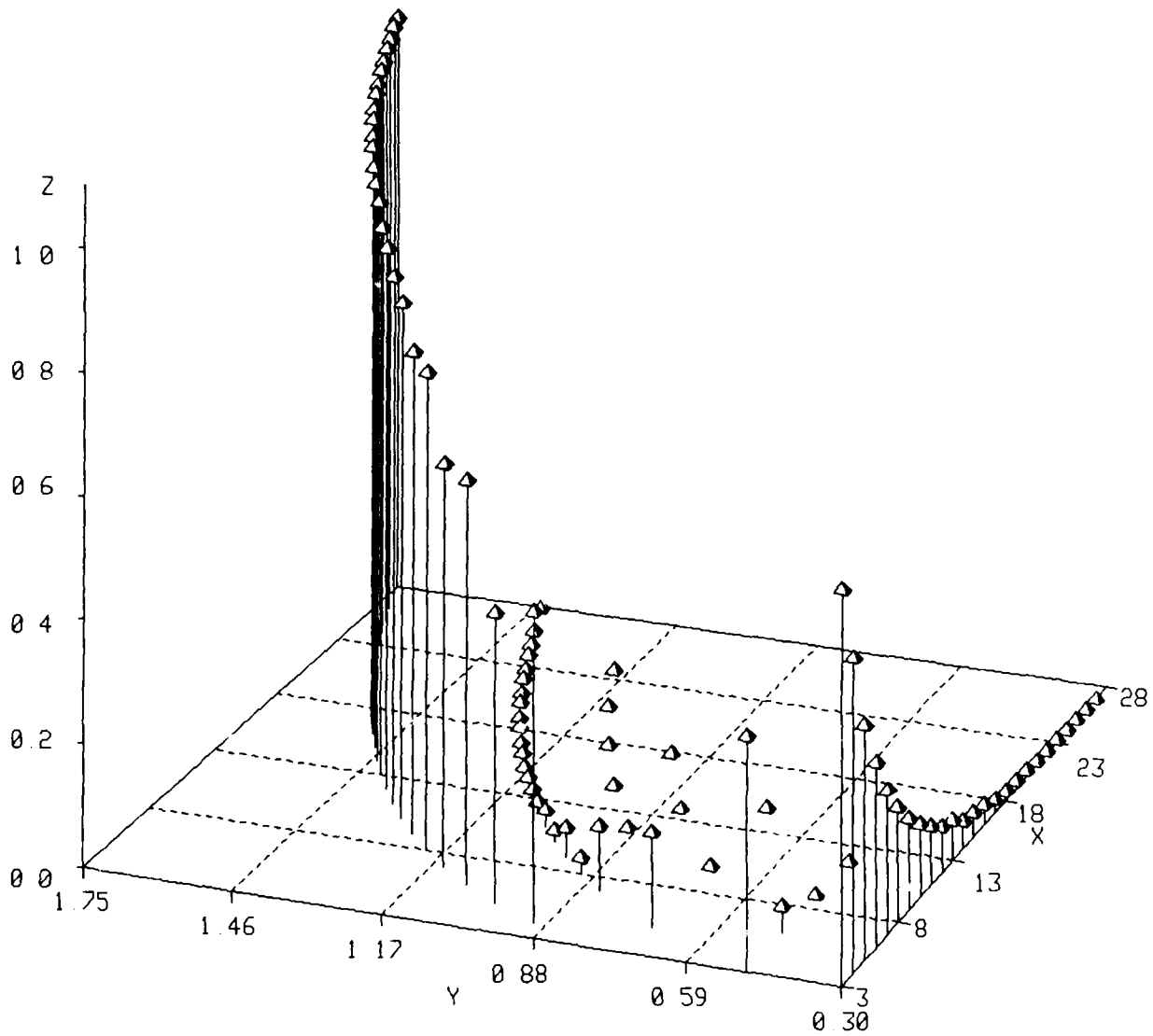


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D1. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 3

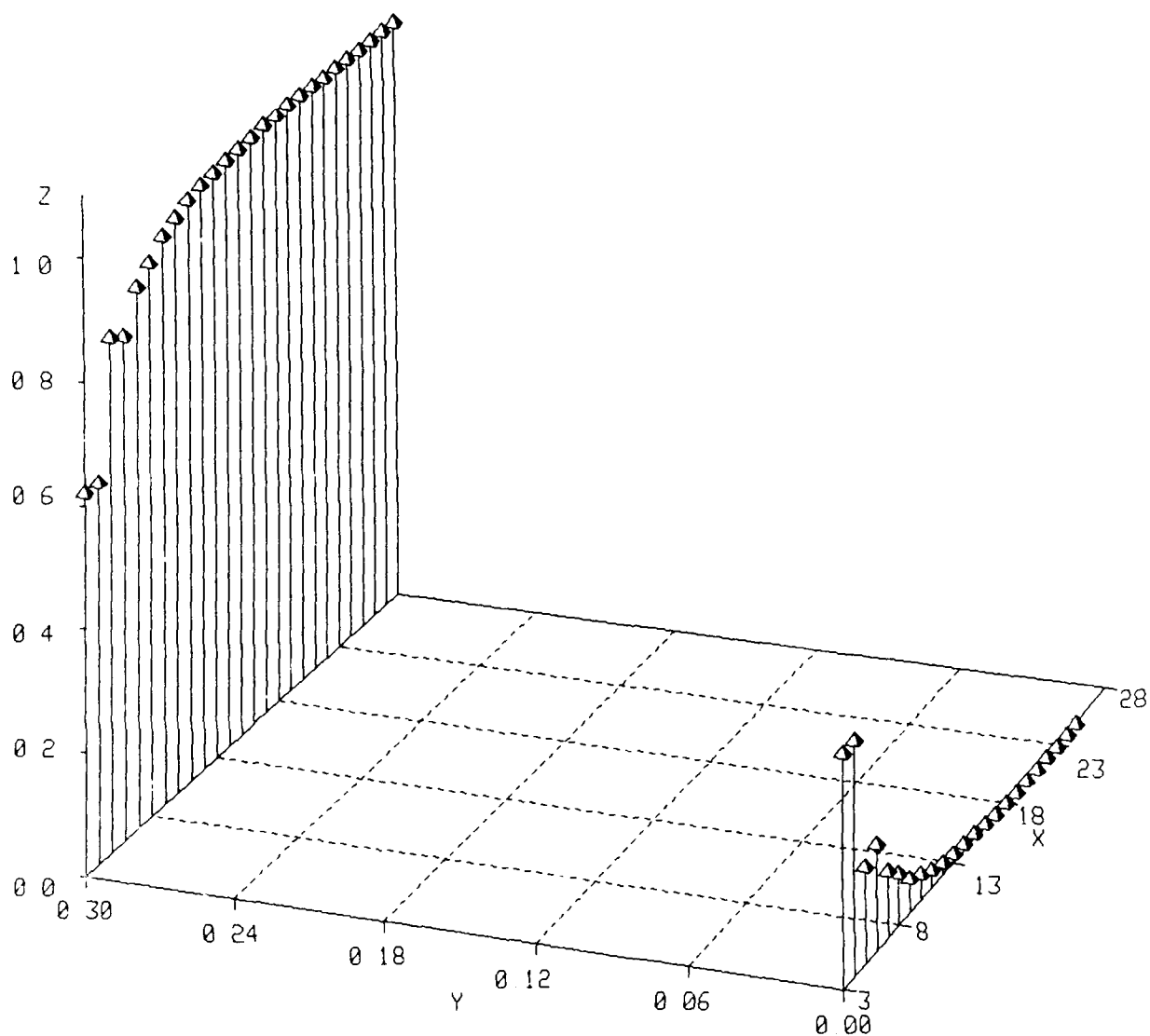


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D2. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 5

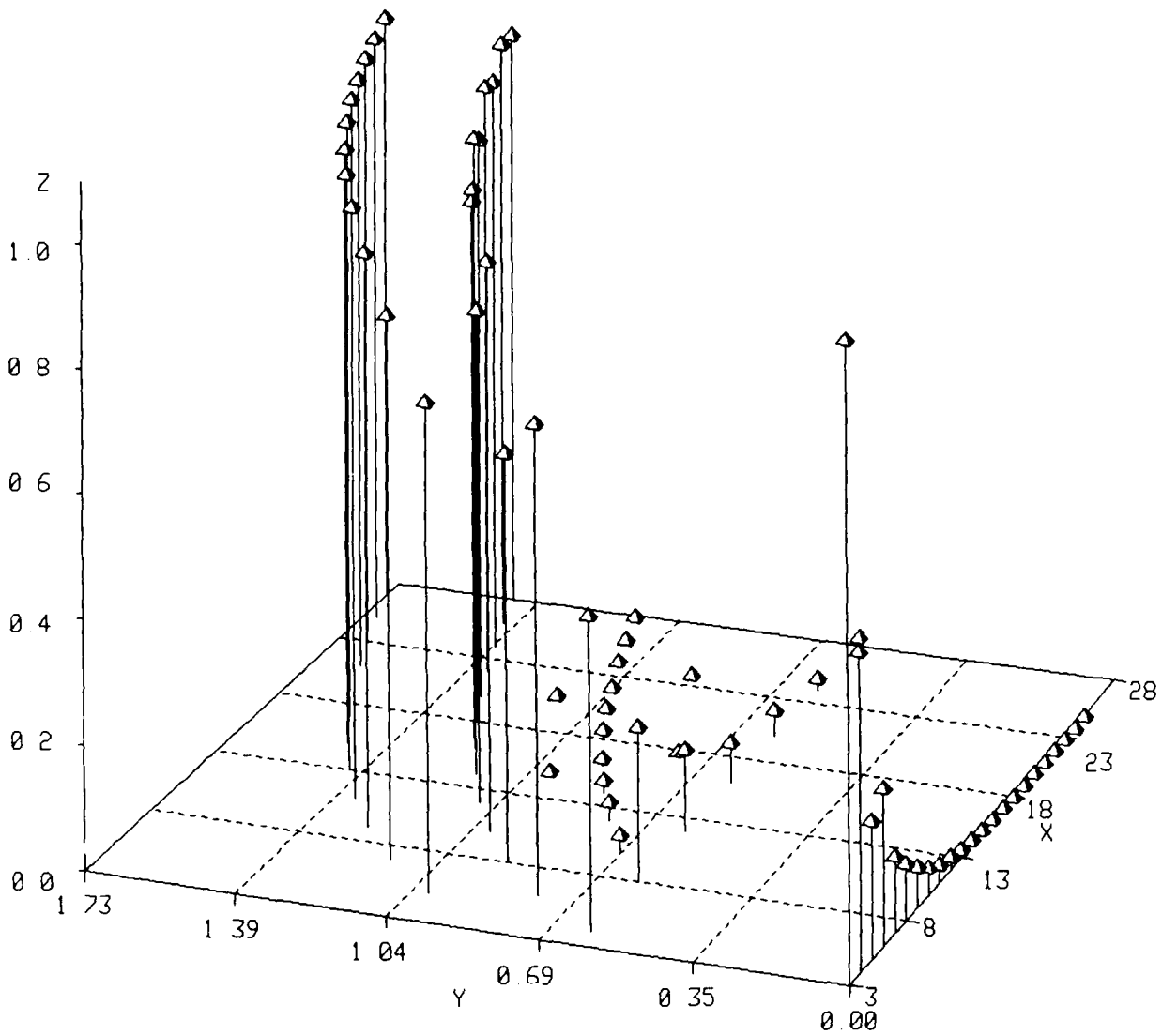


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D3. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 6

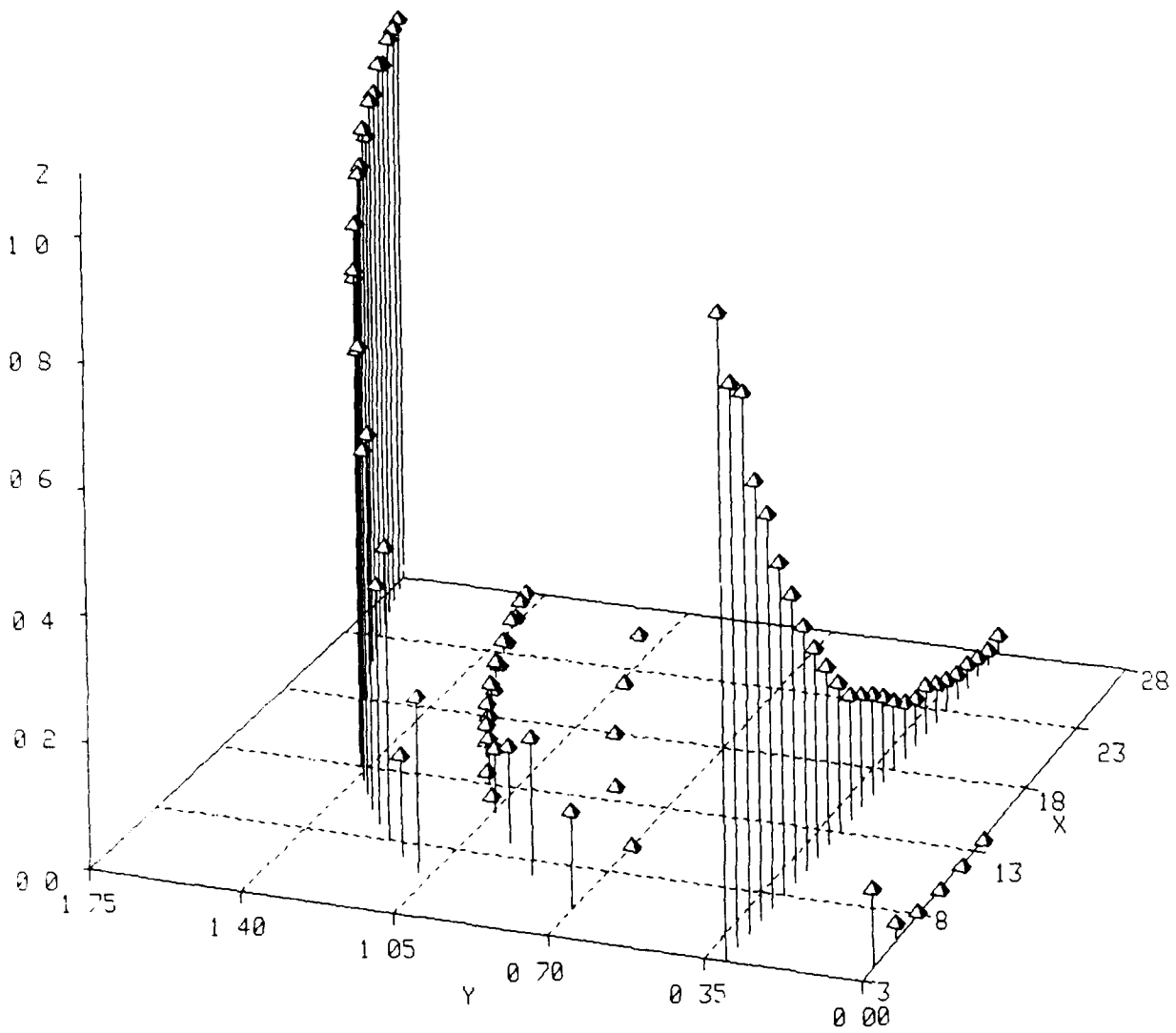


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D4. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 7

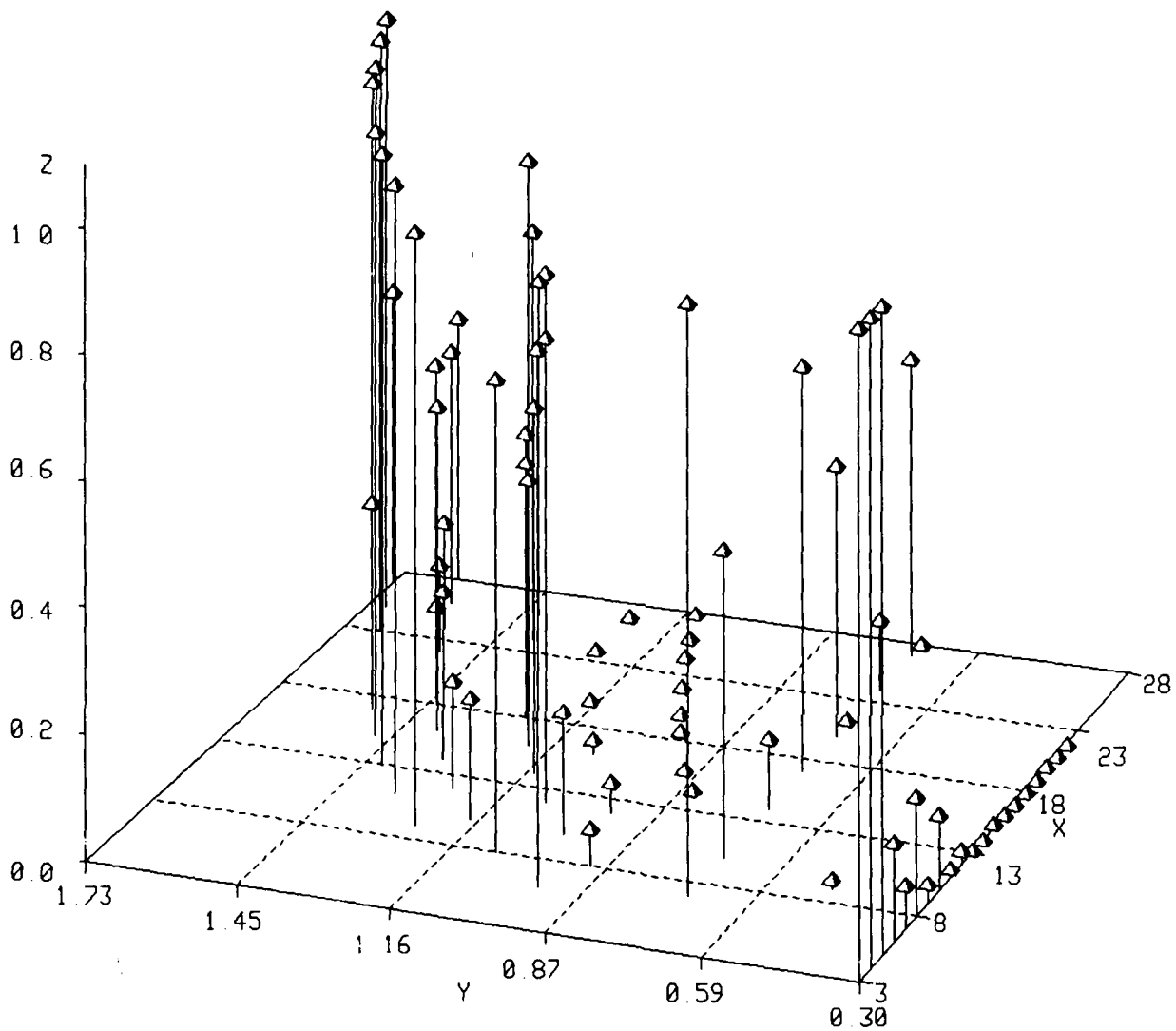


Legend

- X** Number of Cells in Array.
- Y** Log (Limit Cycle Length)
- Z** Fraction of State Space Occupied by Attractor Basin.

Figure D5. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 9

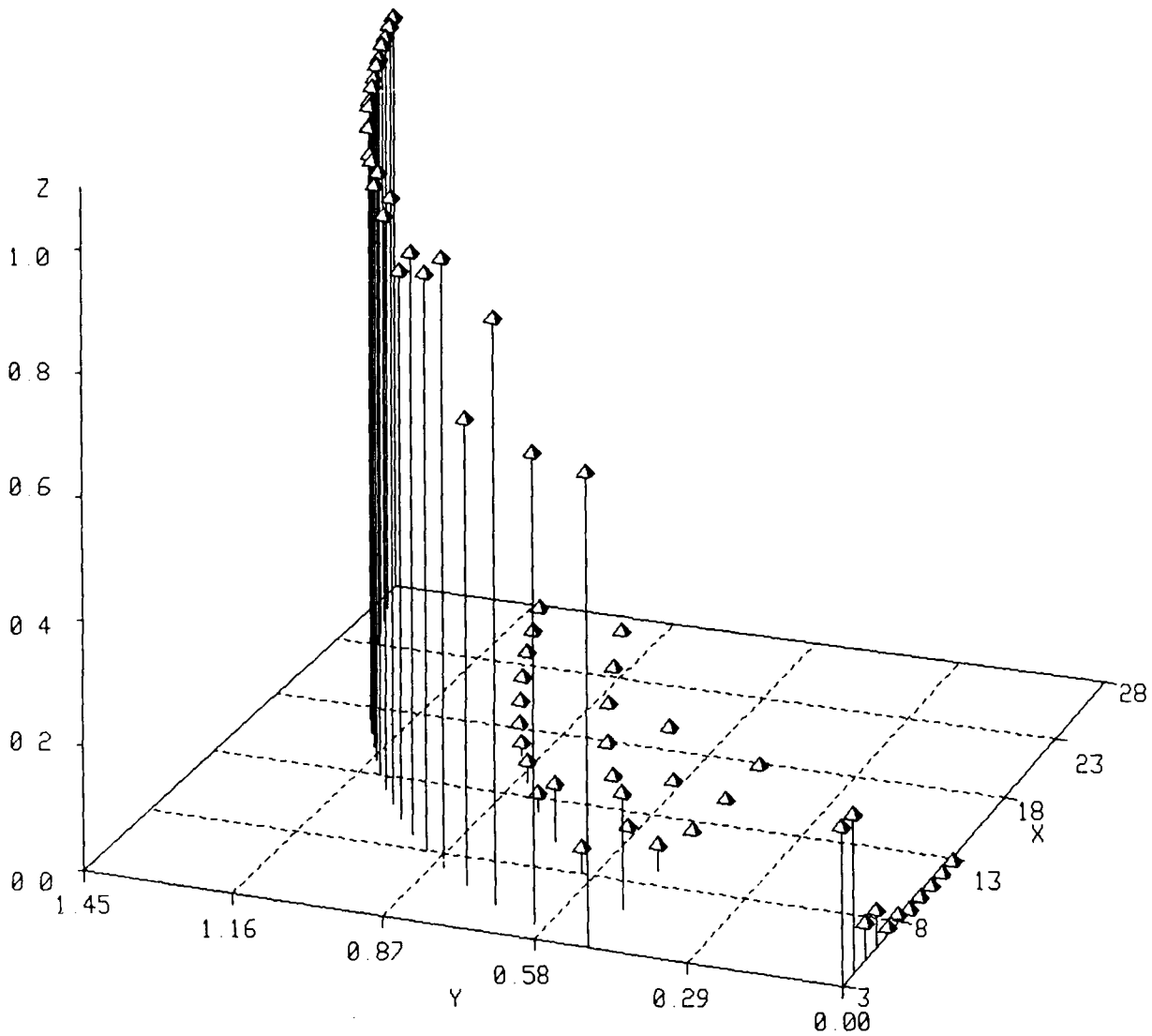


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D6. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 10

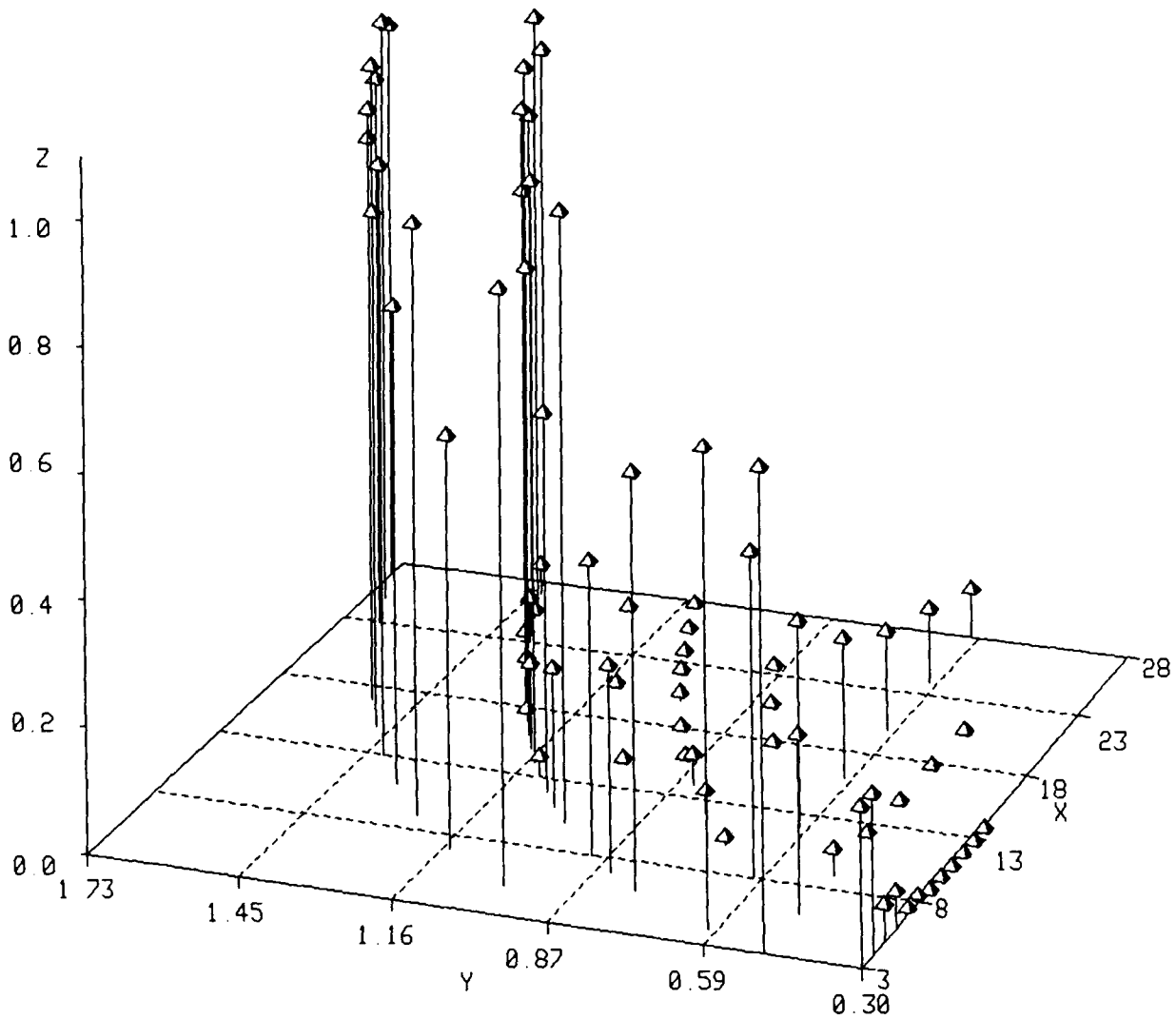


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D7. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 11

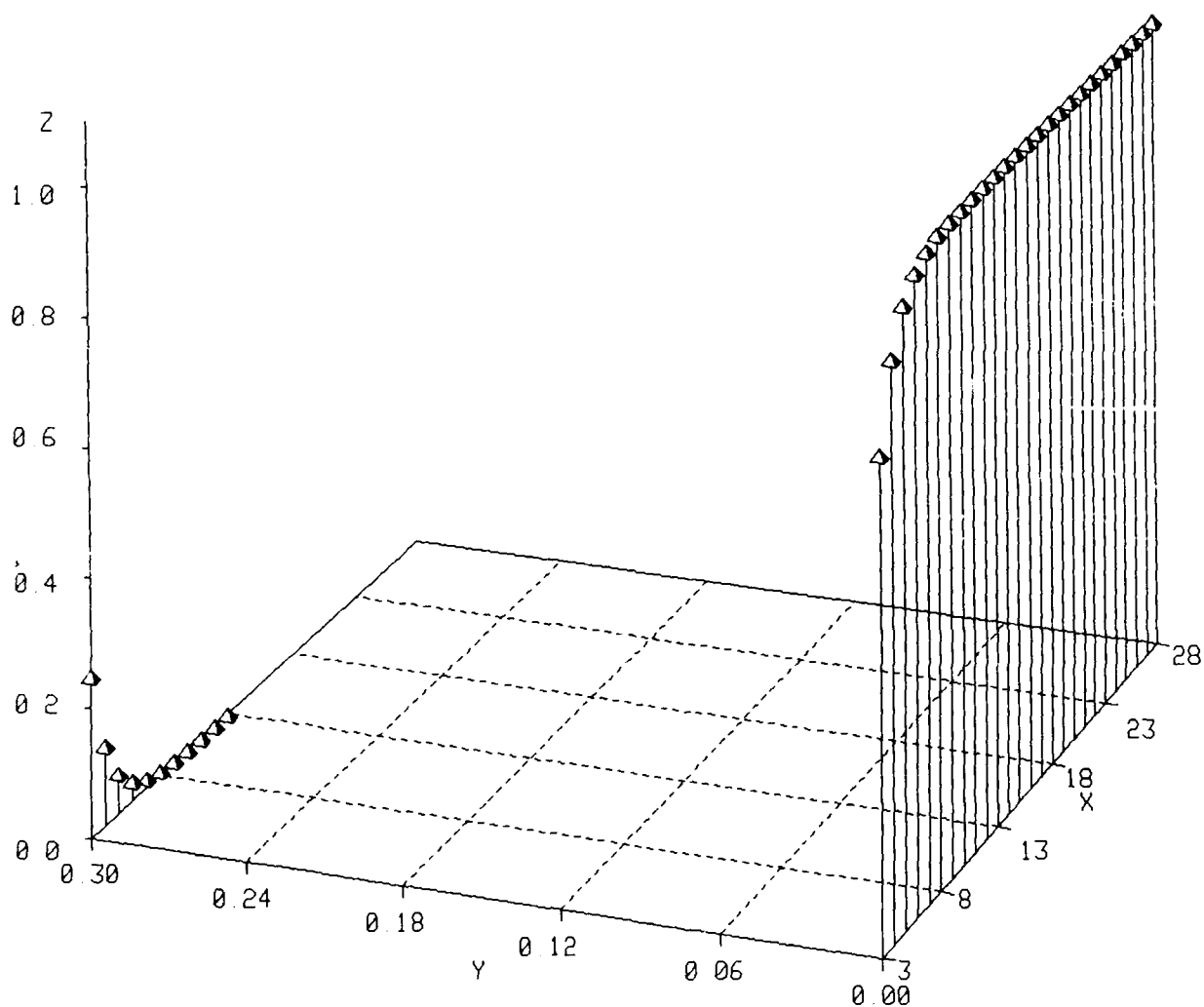


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D8. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 13

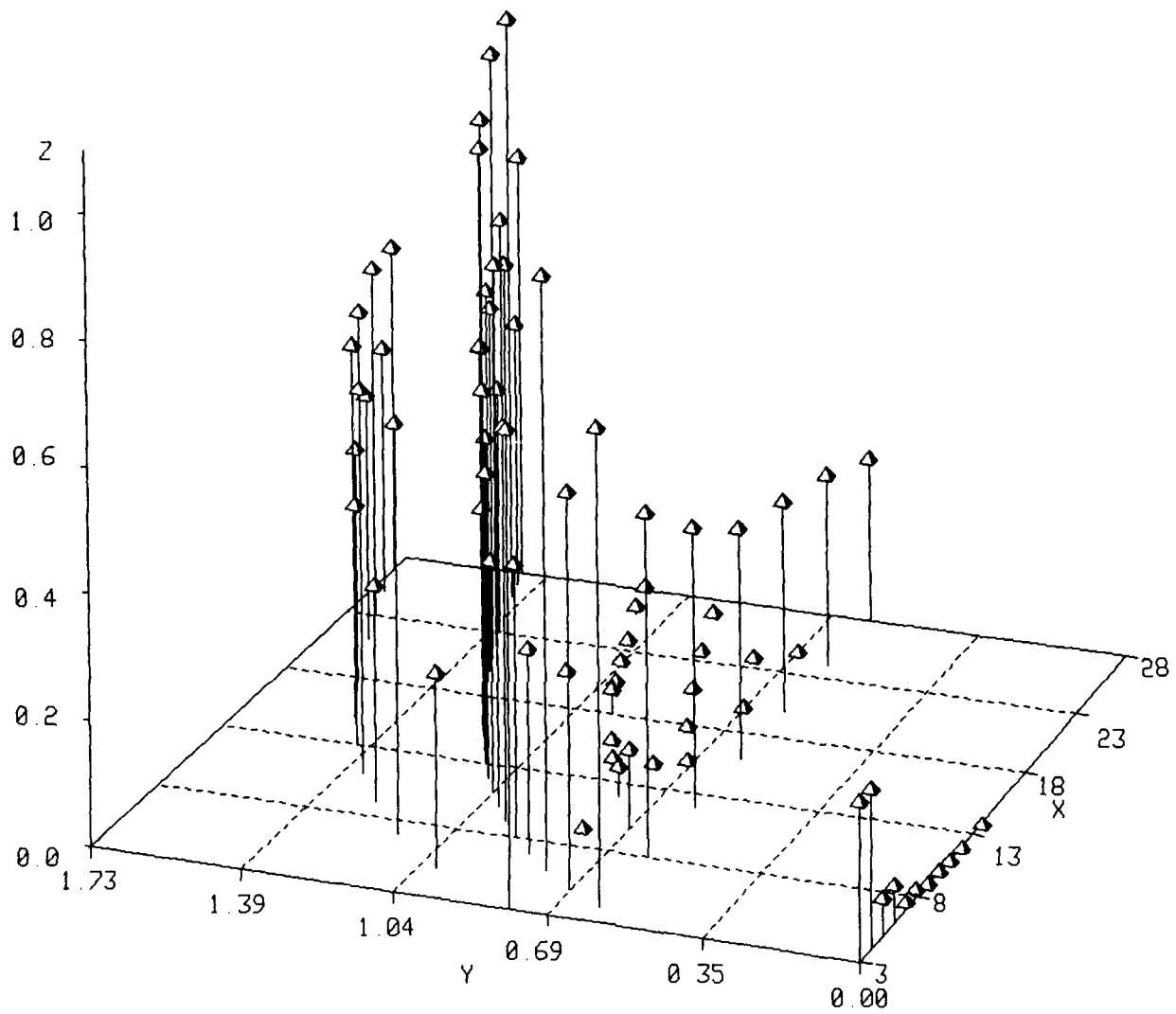


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D9. Attractor probabilities vs array size. Rule is defined in Appendix A.

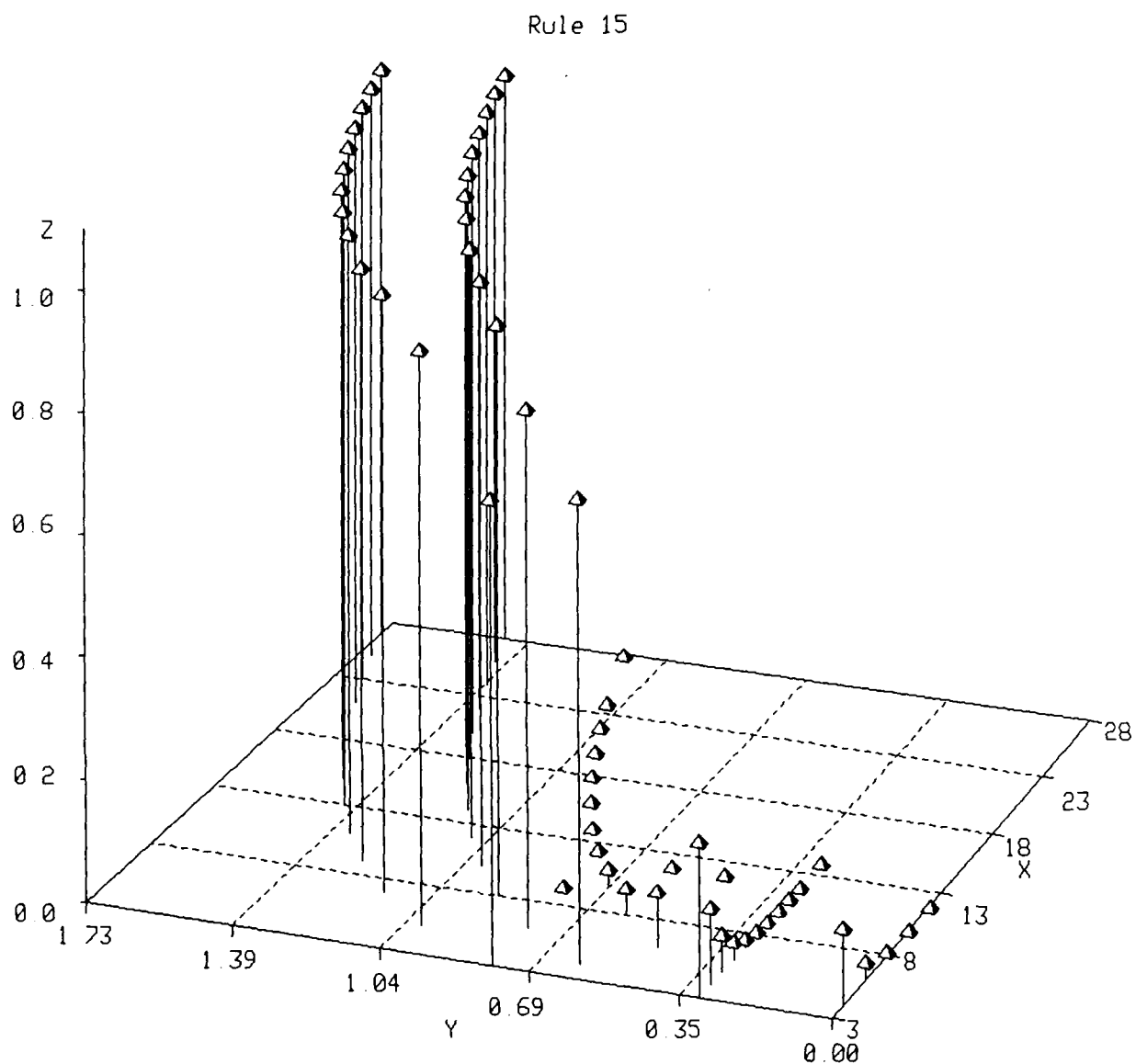
Rule 14



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D10. Attractor probabilities vs array size. Rule is defined in Appendix A.

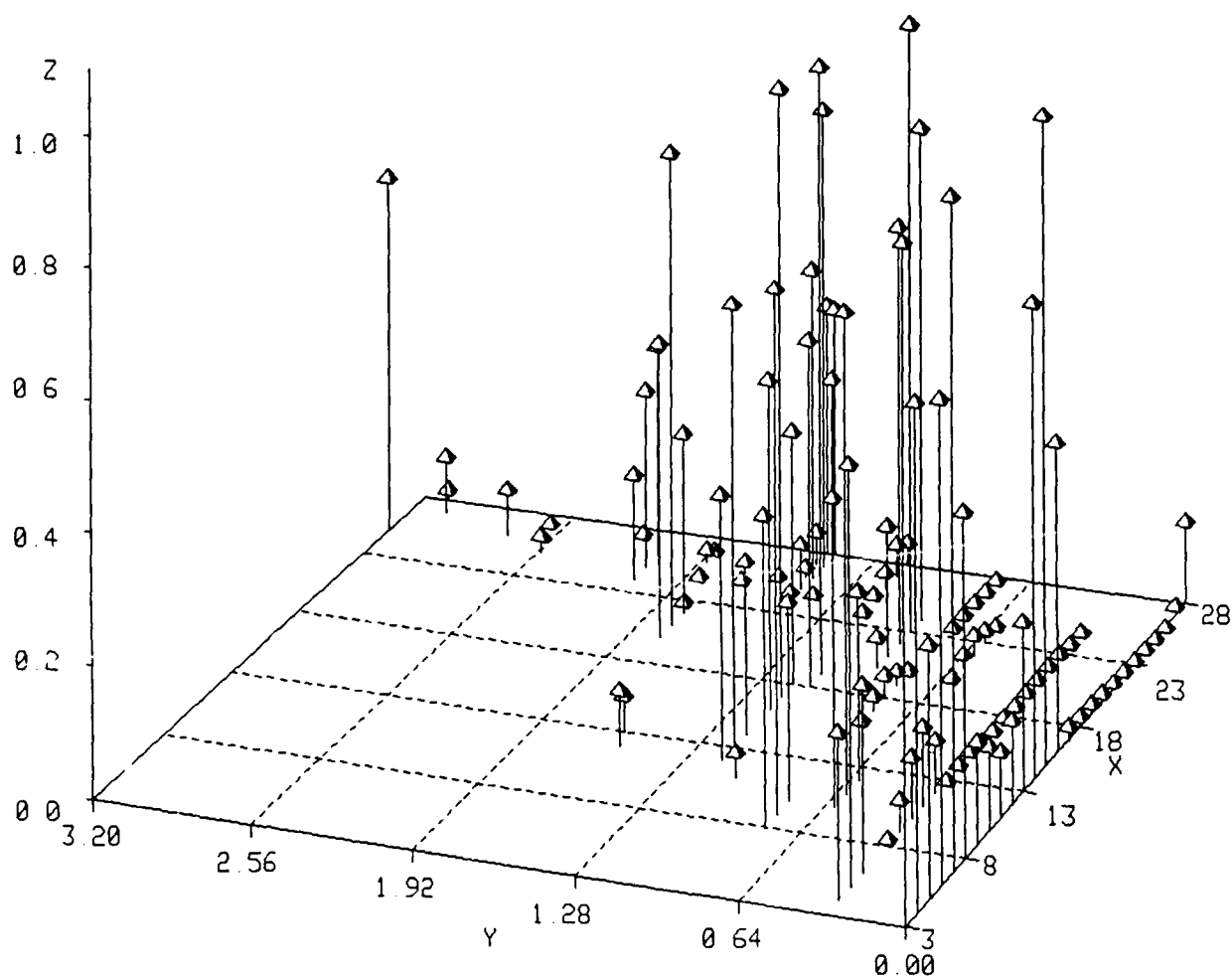


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D11. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 18

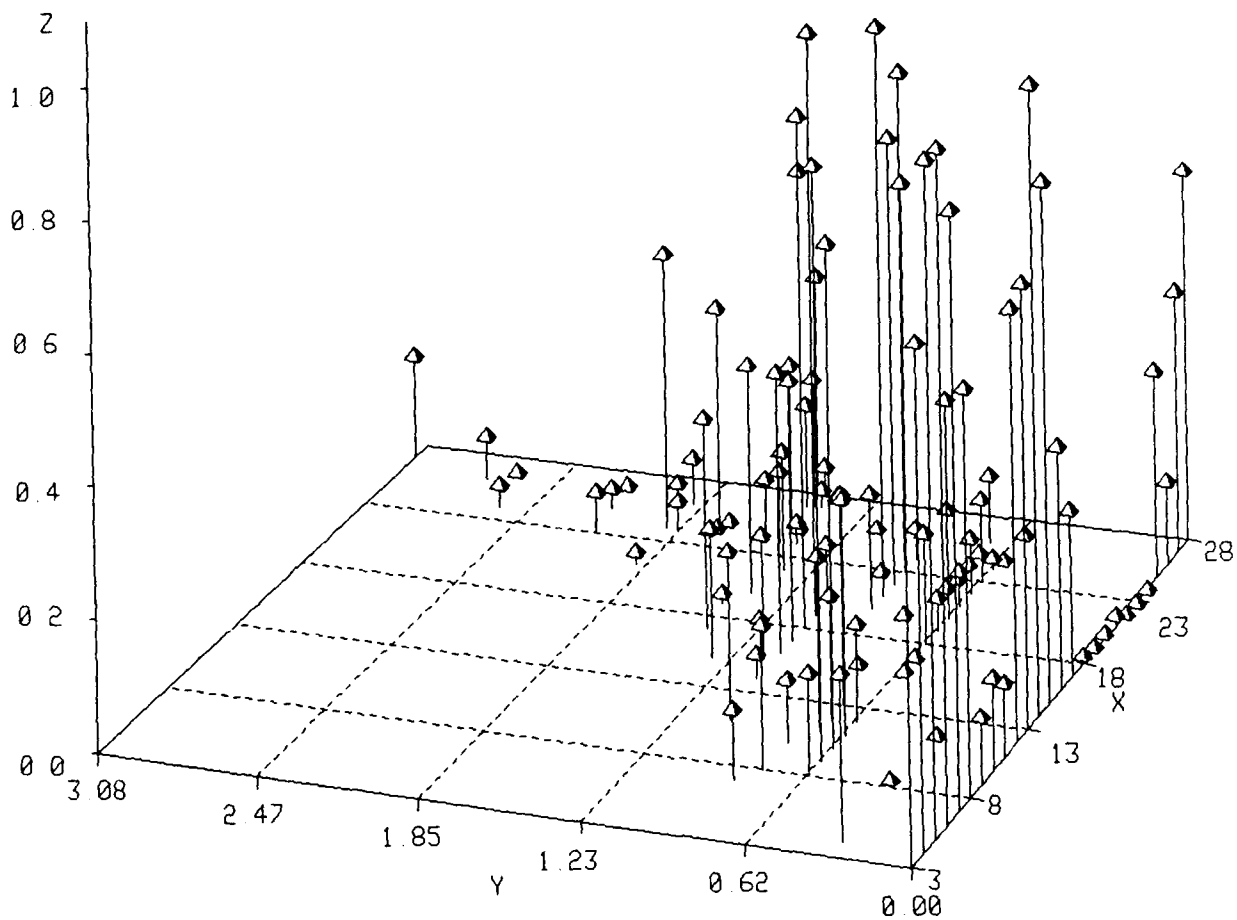


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D12. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 22

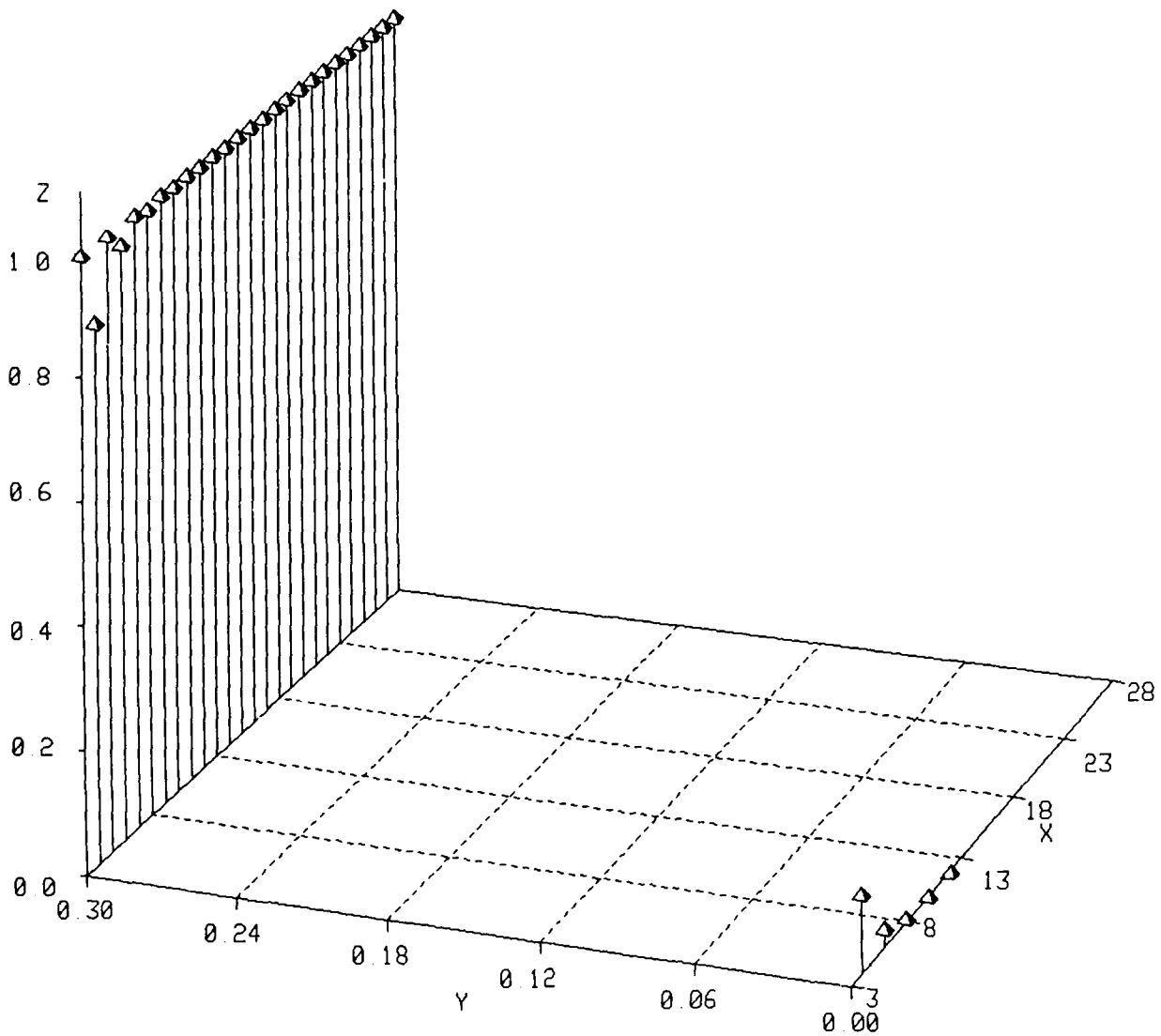


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D13. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 23

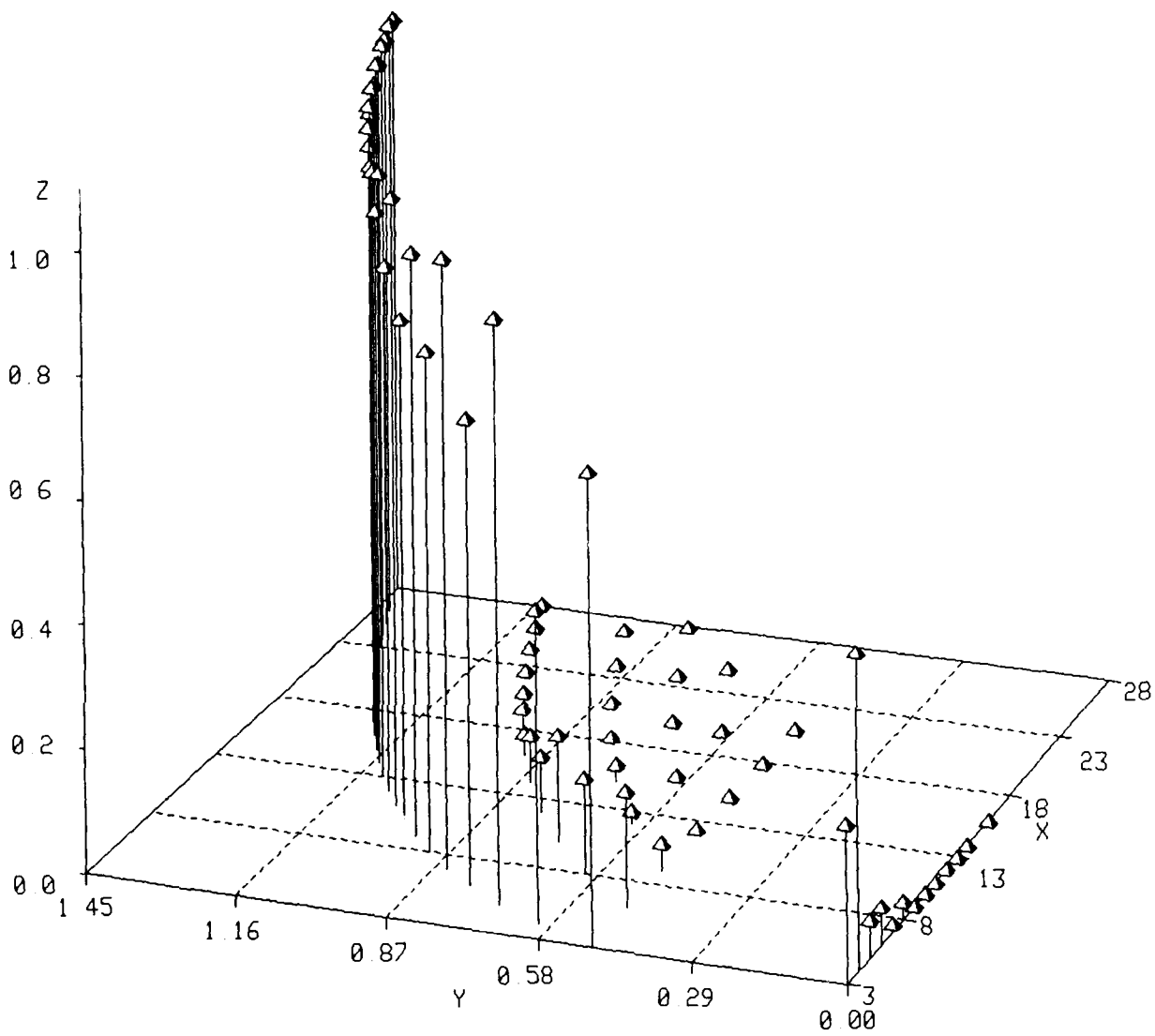


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D14. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 24

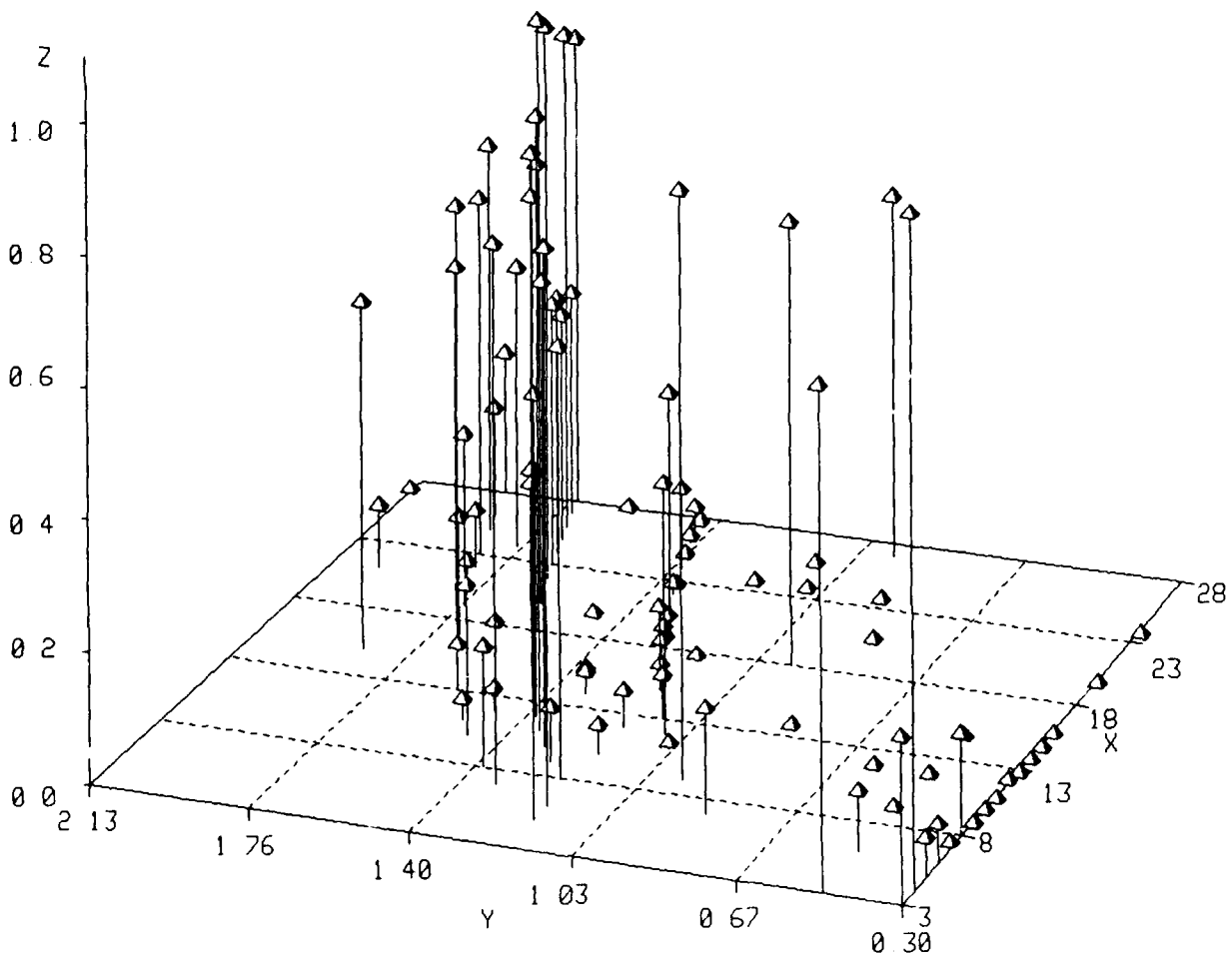


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D15. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 25

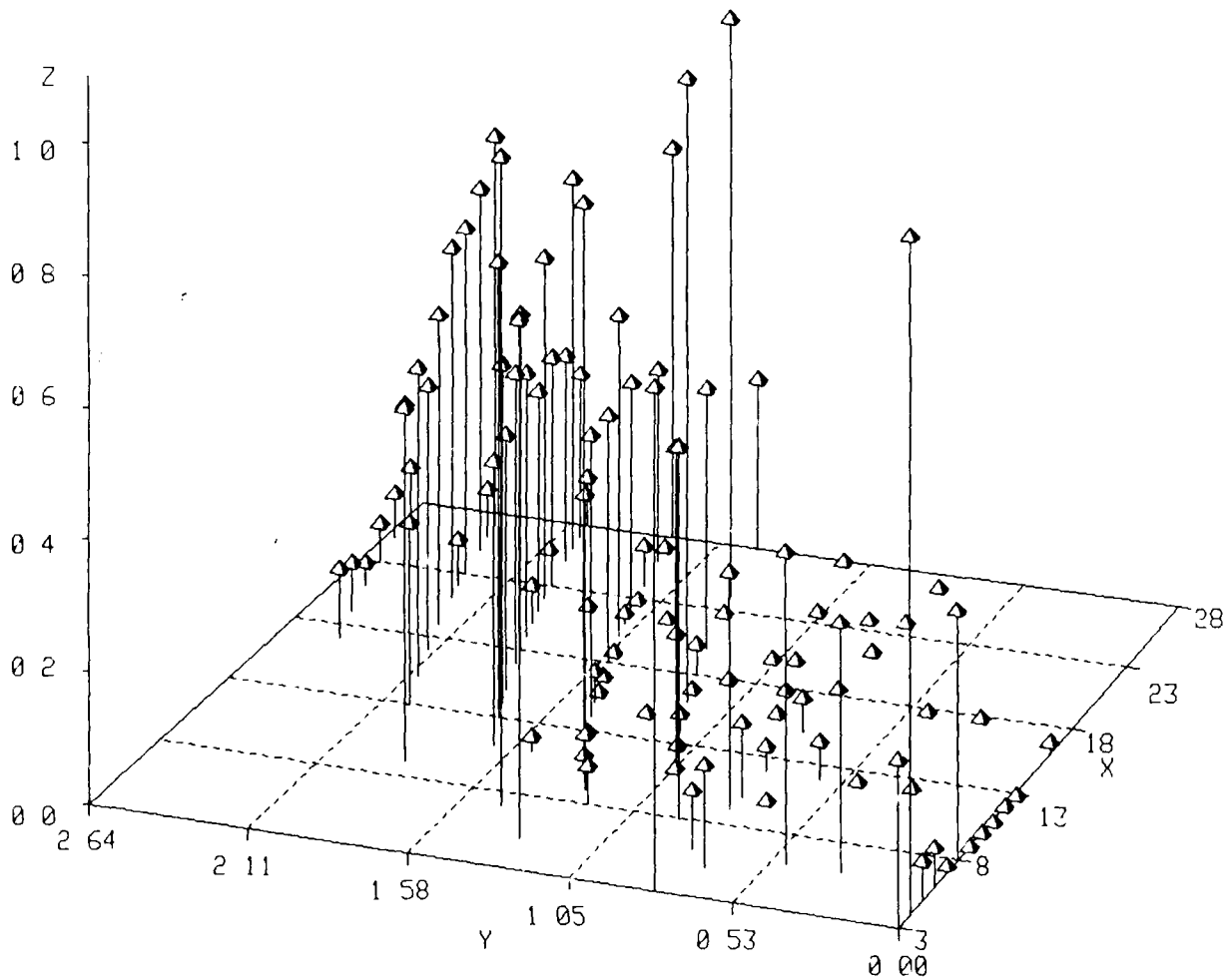


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D16. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 26

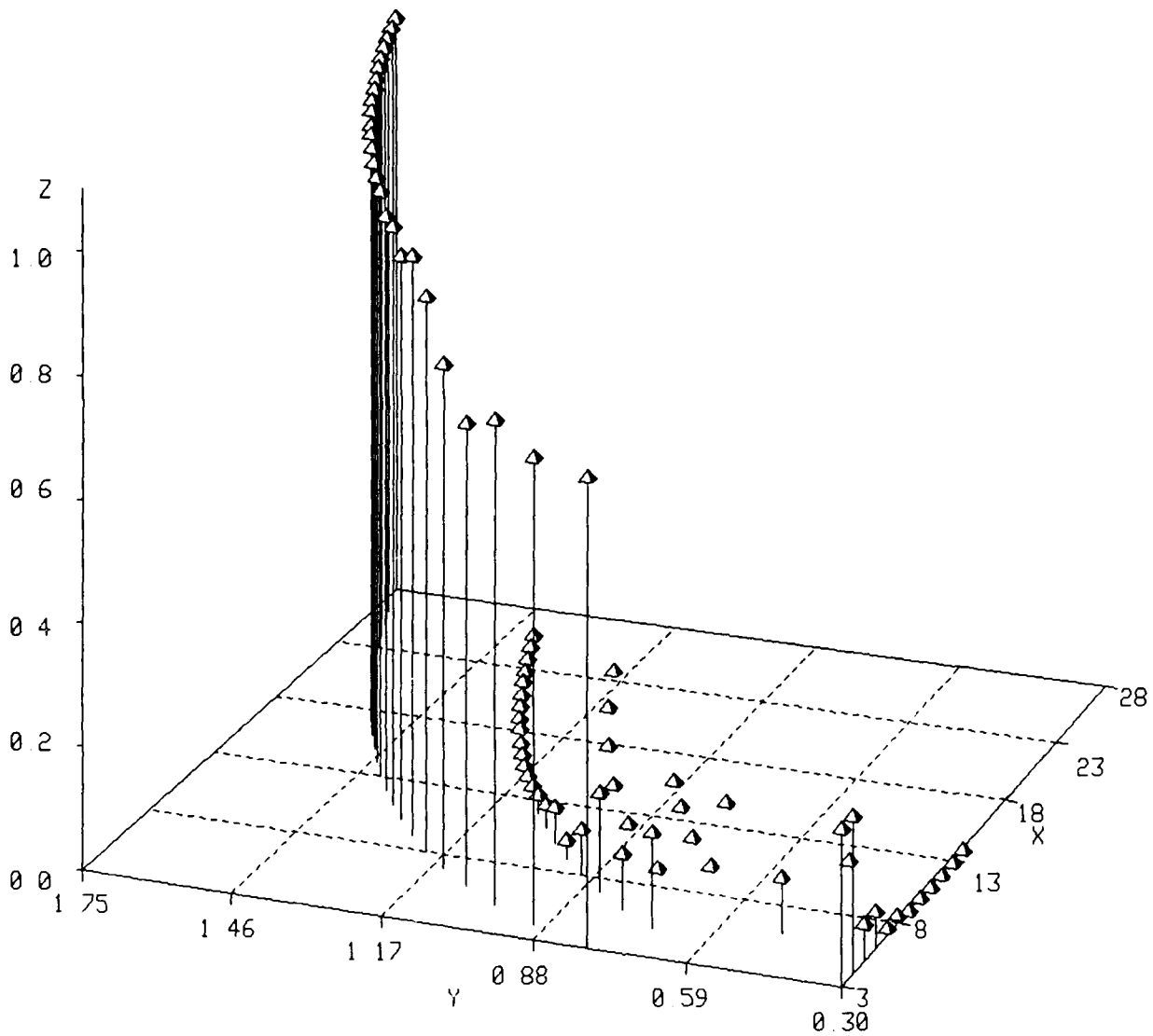


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D17. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 27

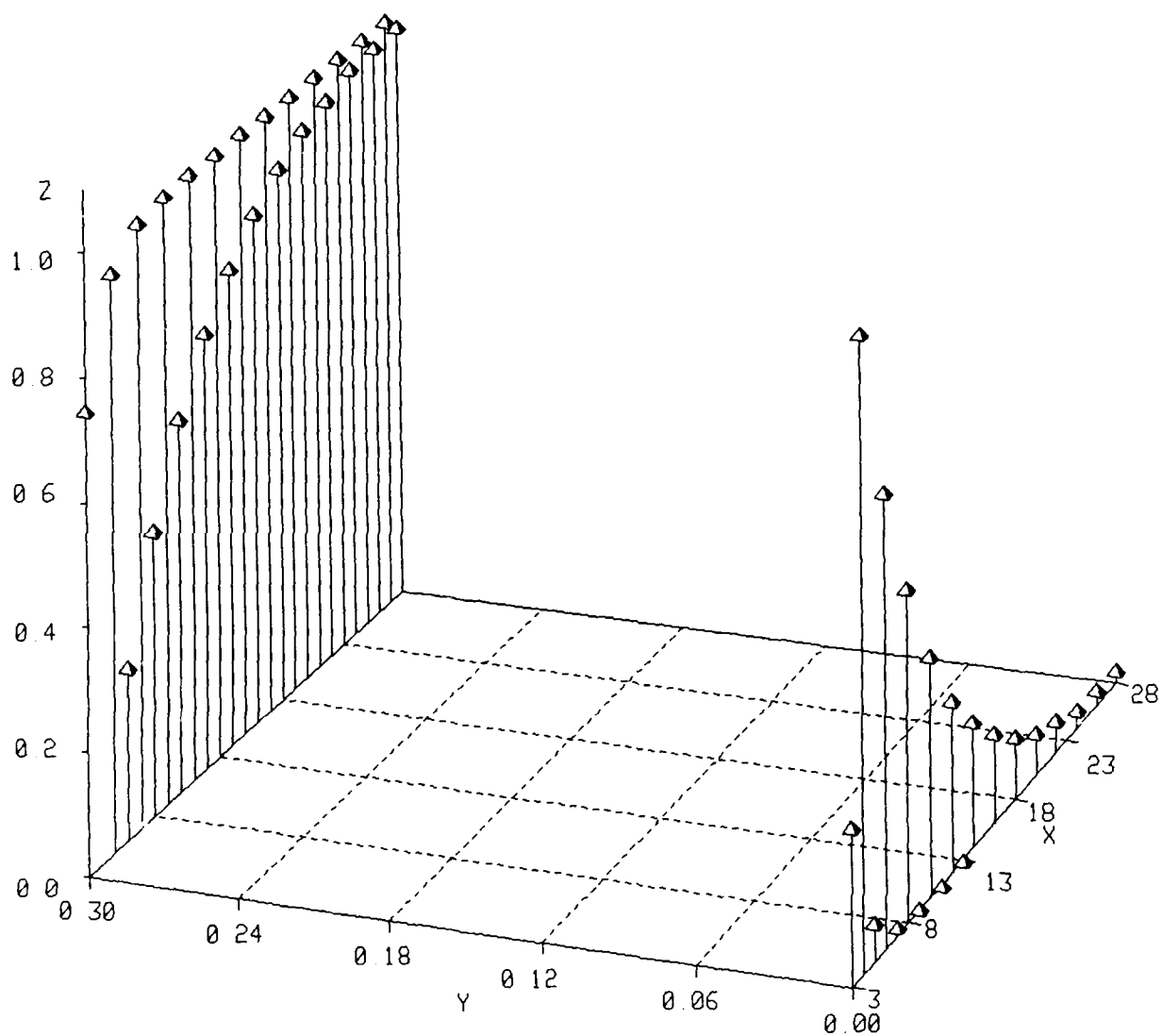


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D18. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 28

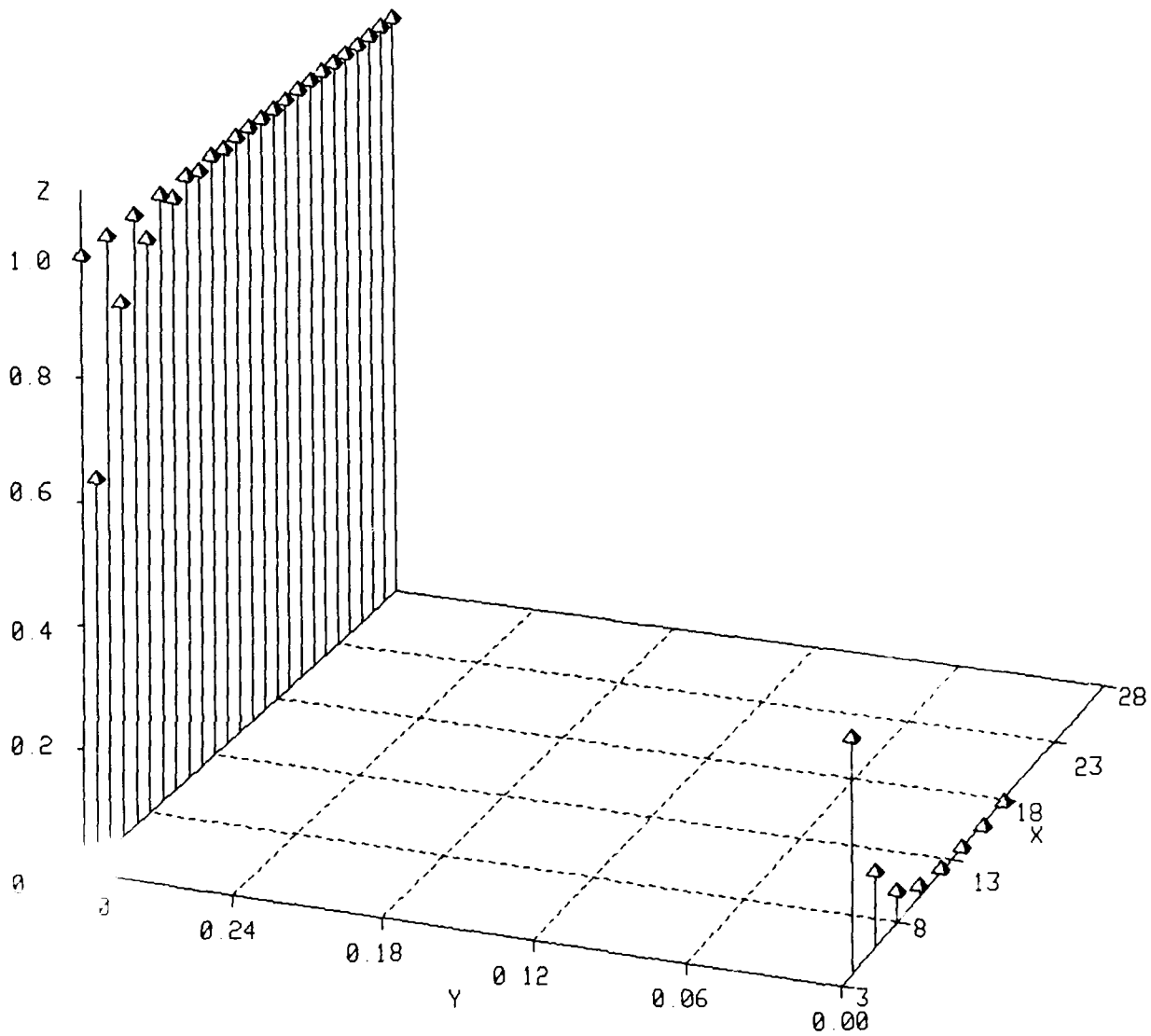


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D19. Attractor probabilities vs array size. Rule is defined in Appendix A.

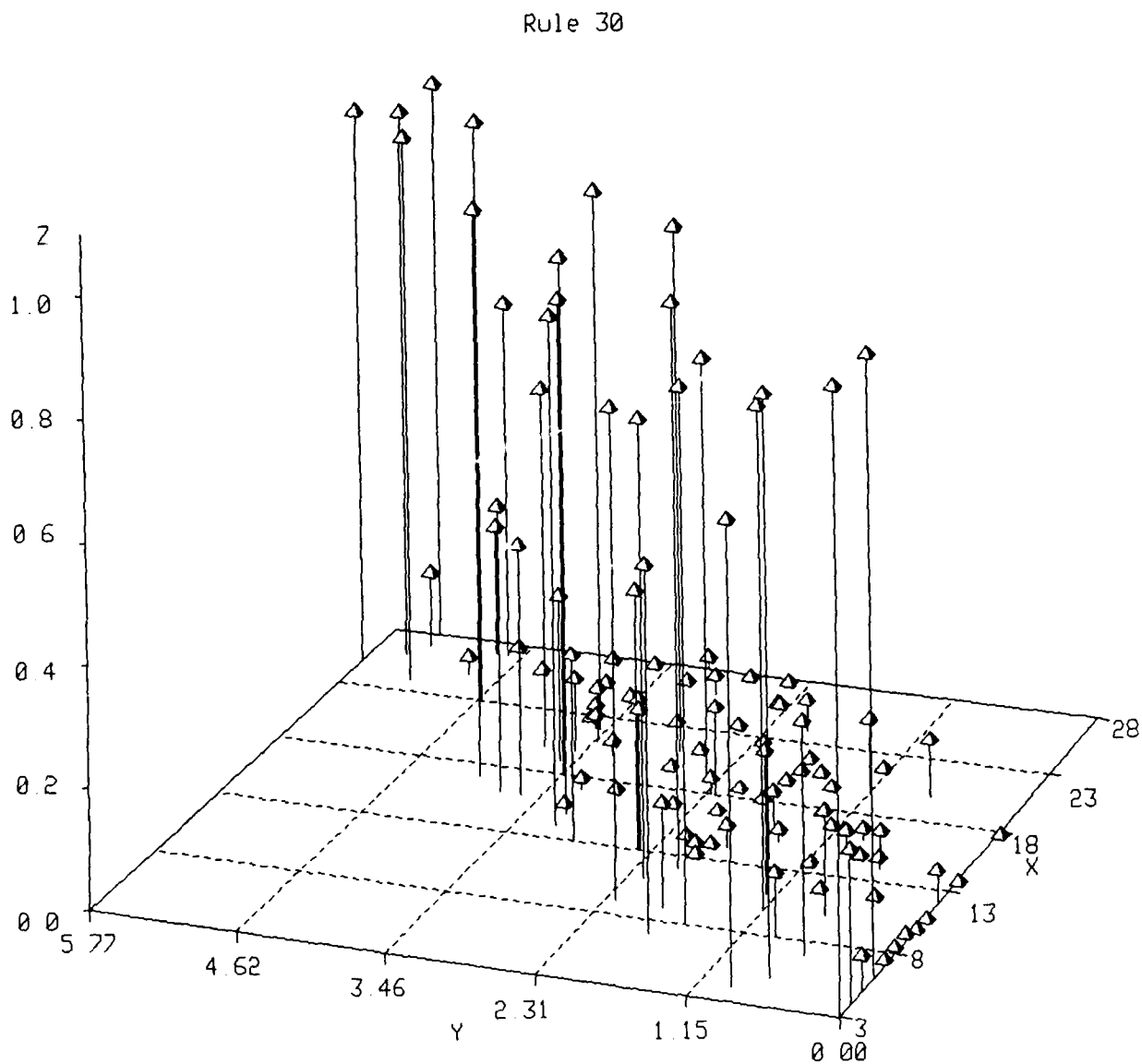
Rule 29



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D20. Attractor probabilities vs array size. Rule is defined in Appendix A.

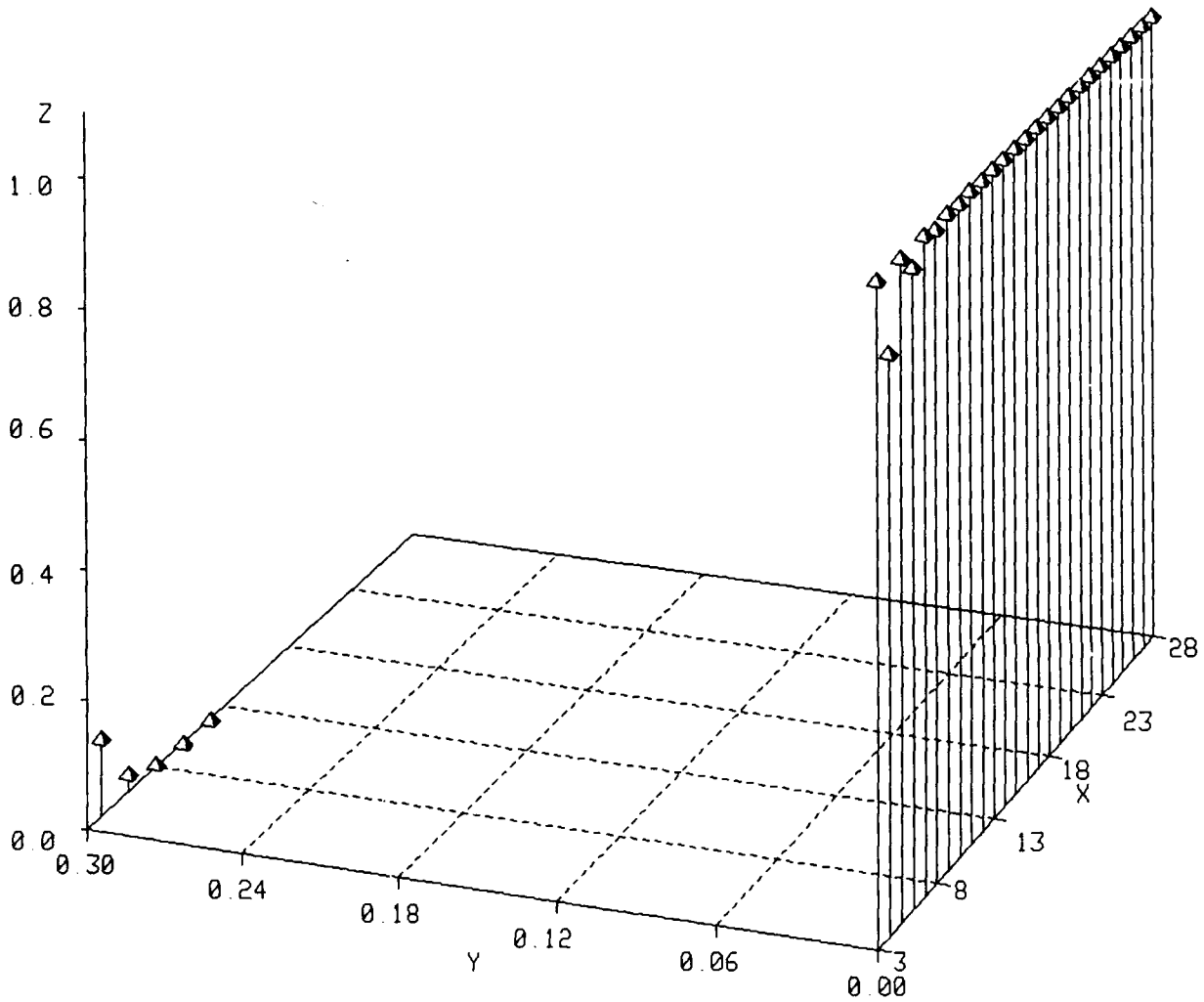


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D21. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 32

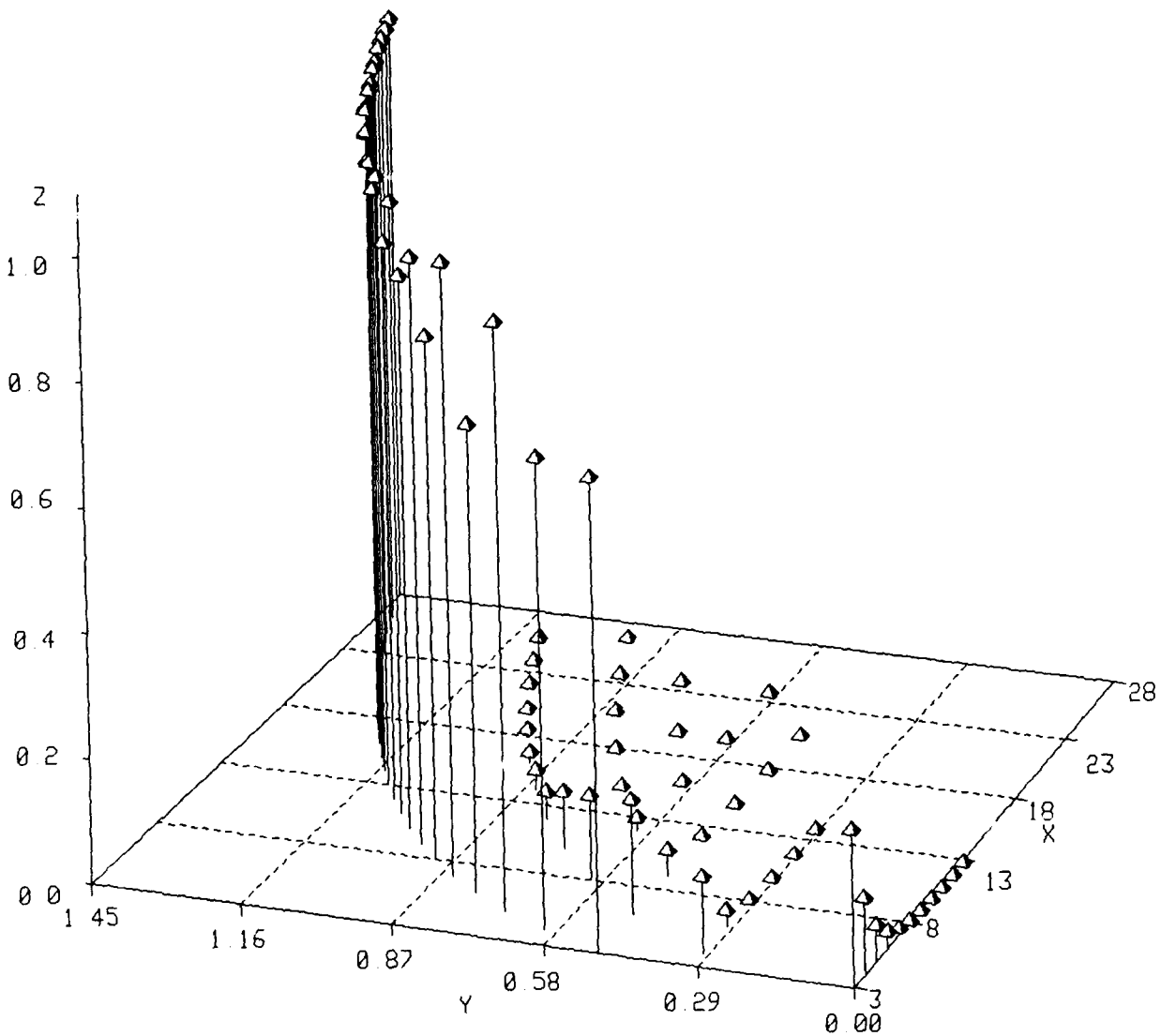


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D22. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 34

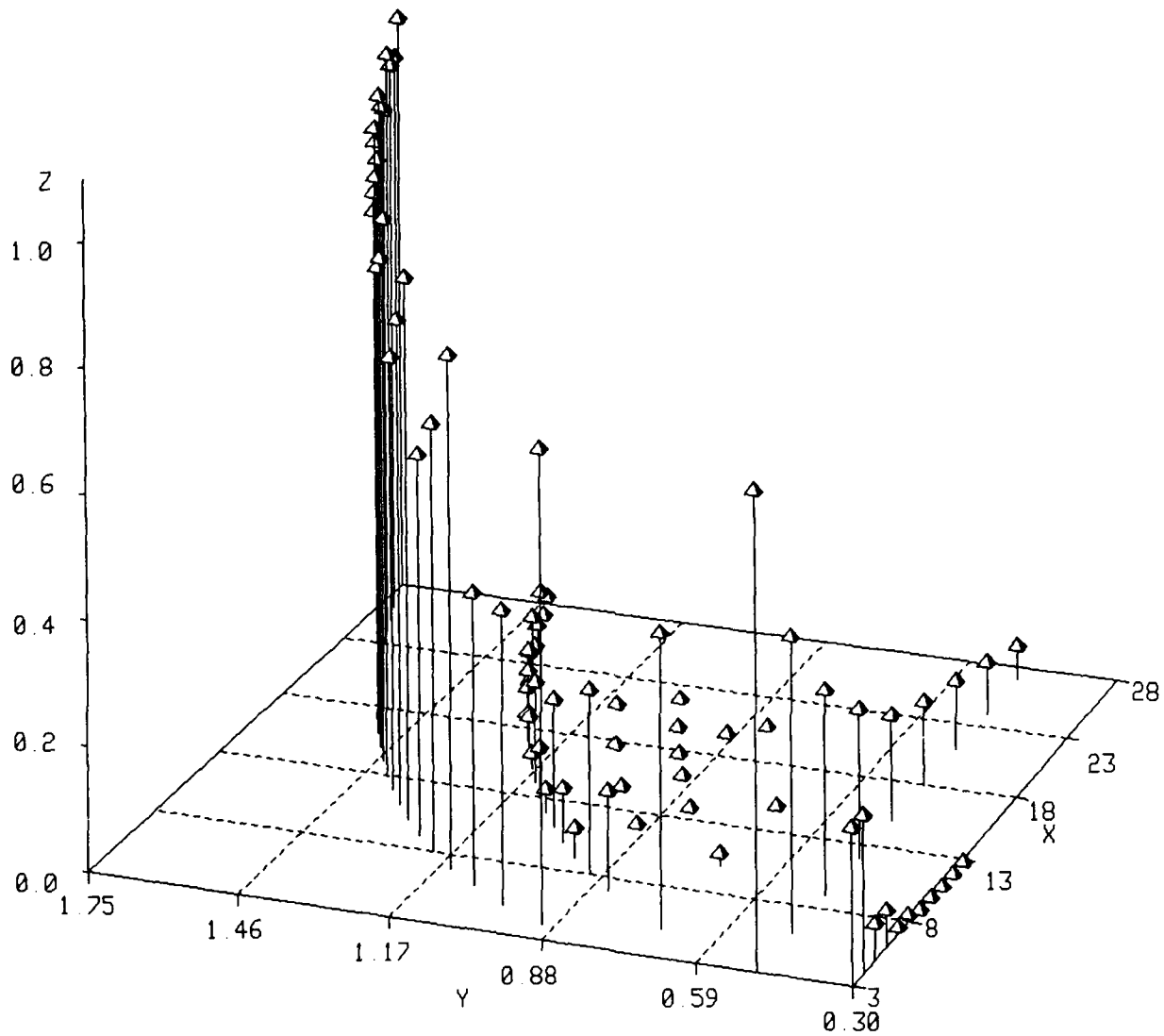


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D23. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 35

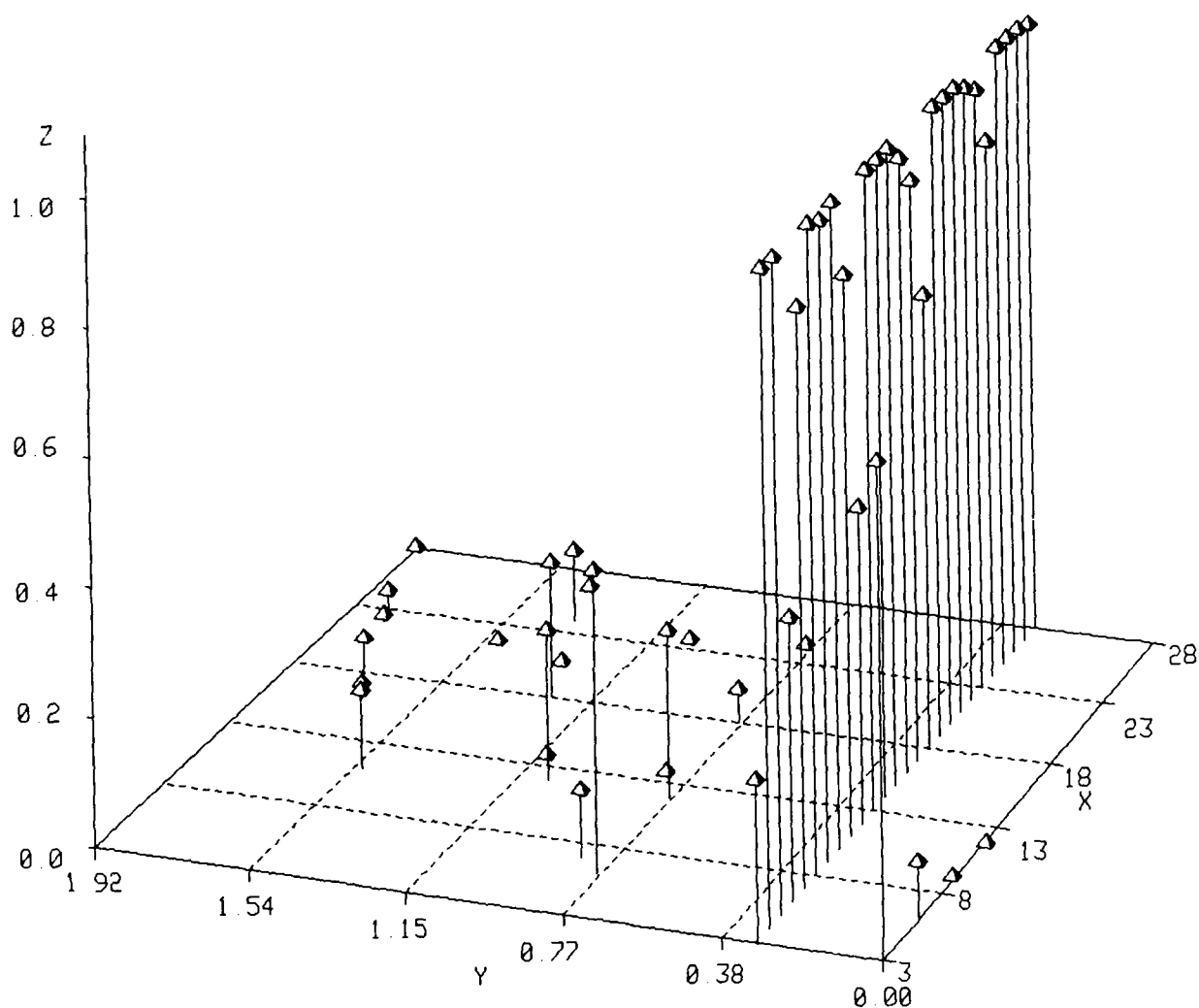


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D24. Attractor probabilities vs array size. Rule is defined in Appendix A.

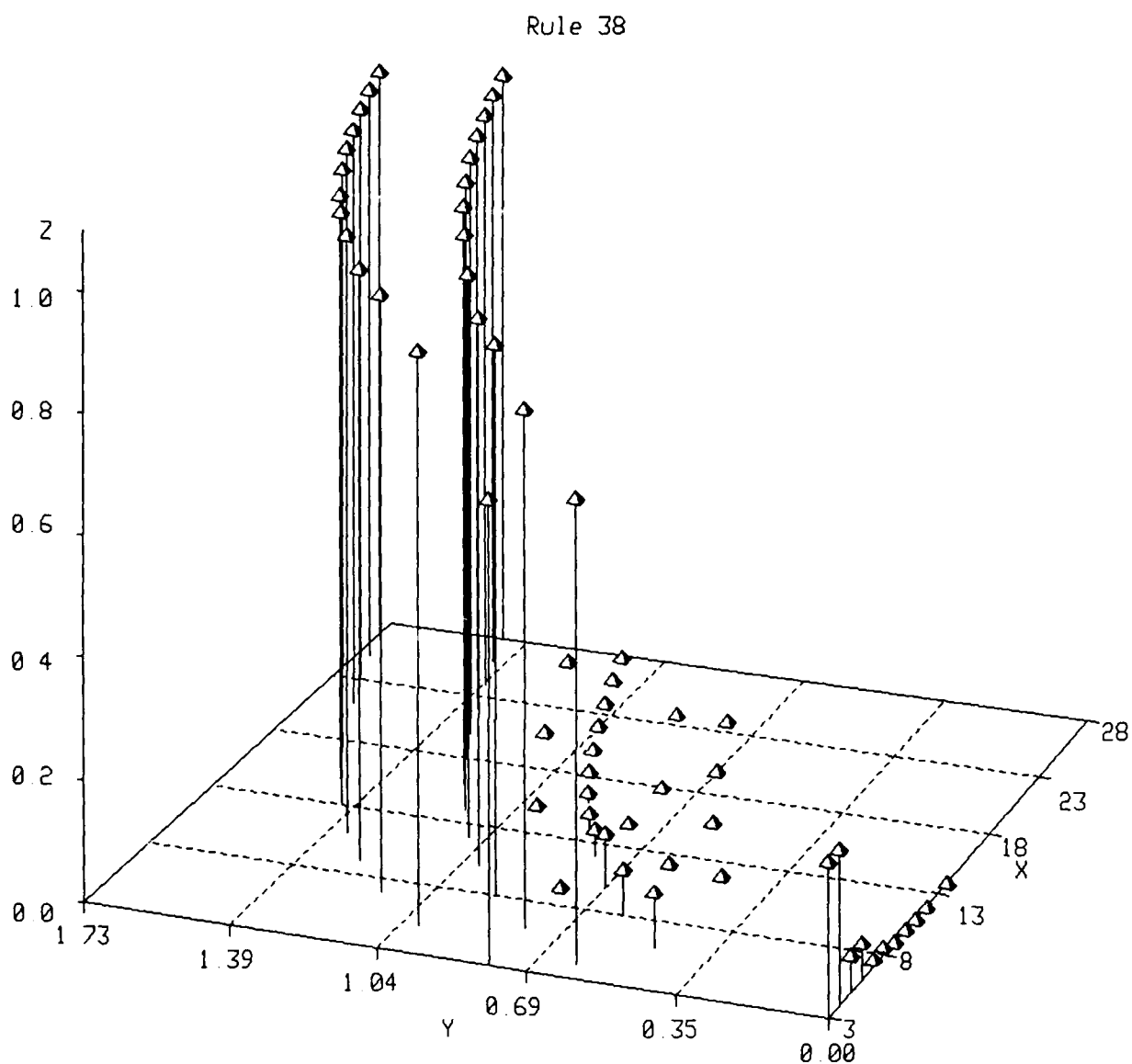
Rule 37



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D25. Attractor probabilities vs array size. Rule is defined in Appendix A.

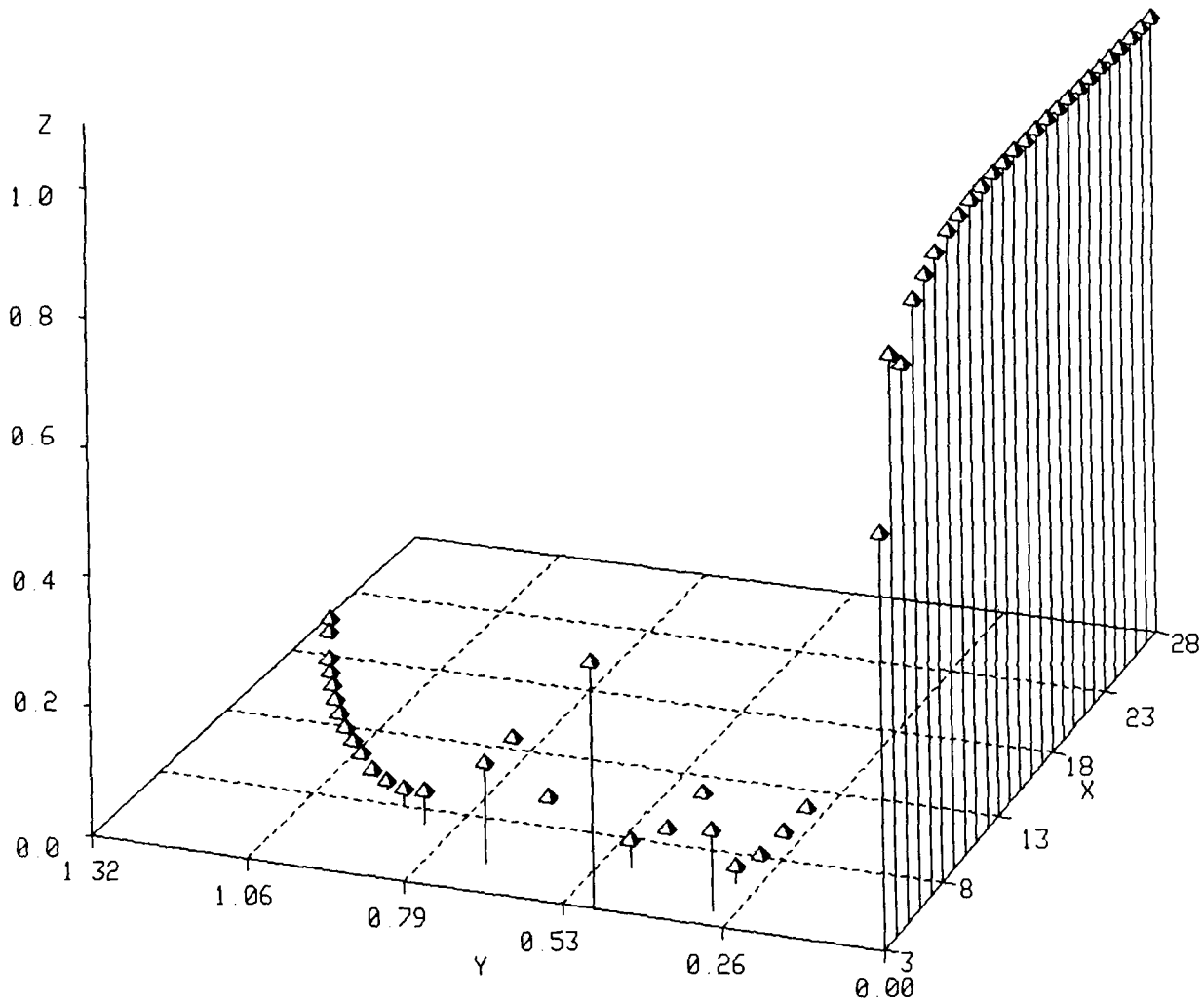


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D26. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 40

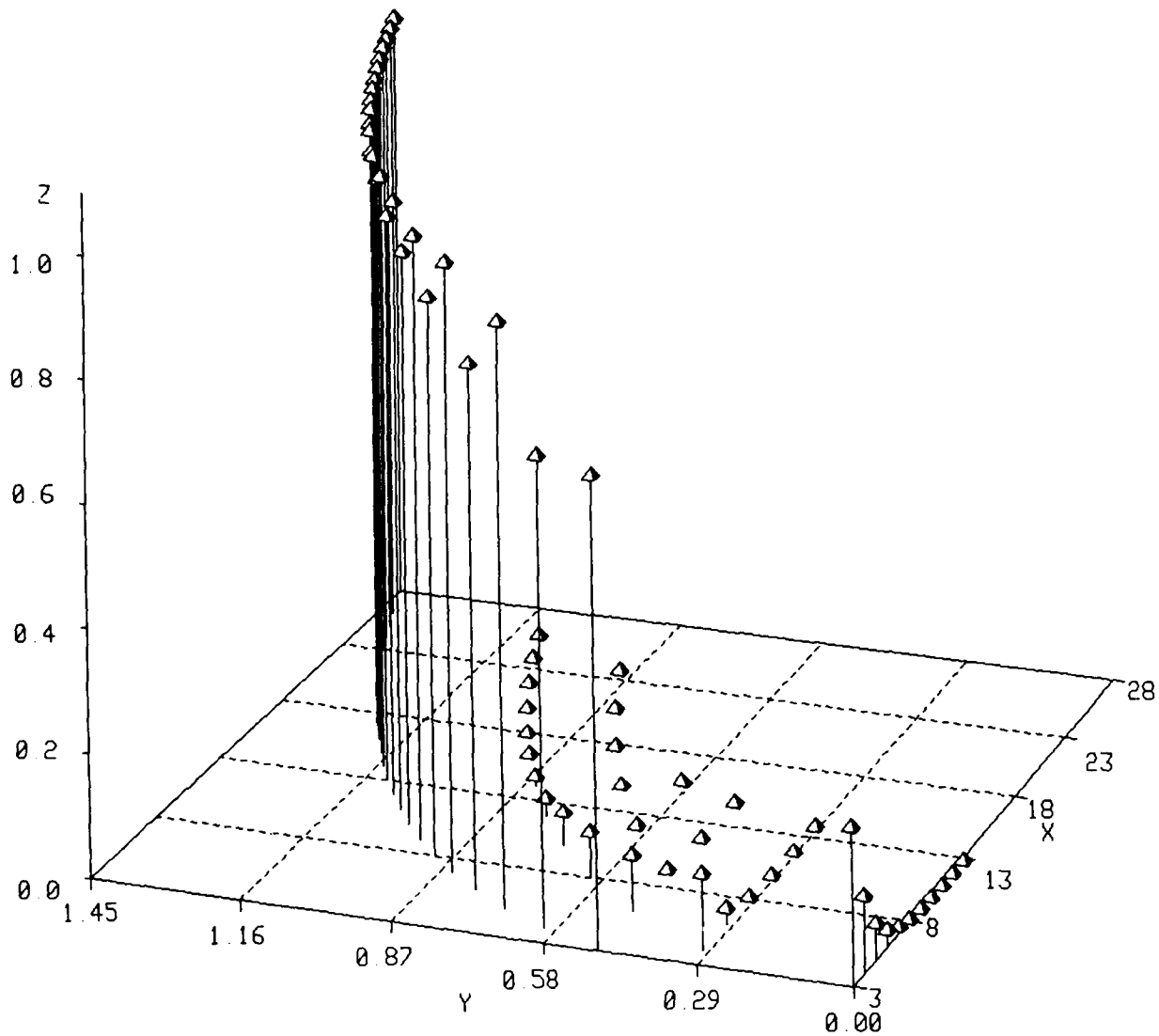


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D27. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 42

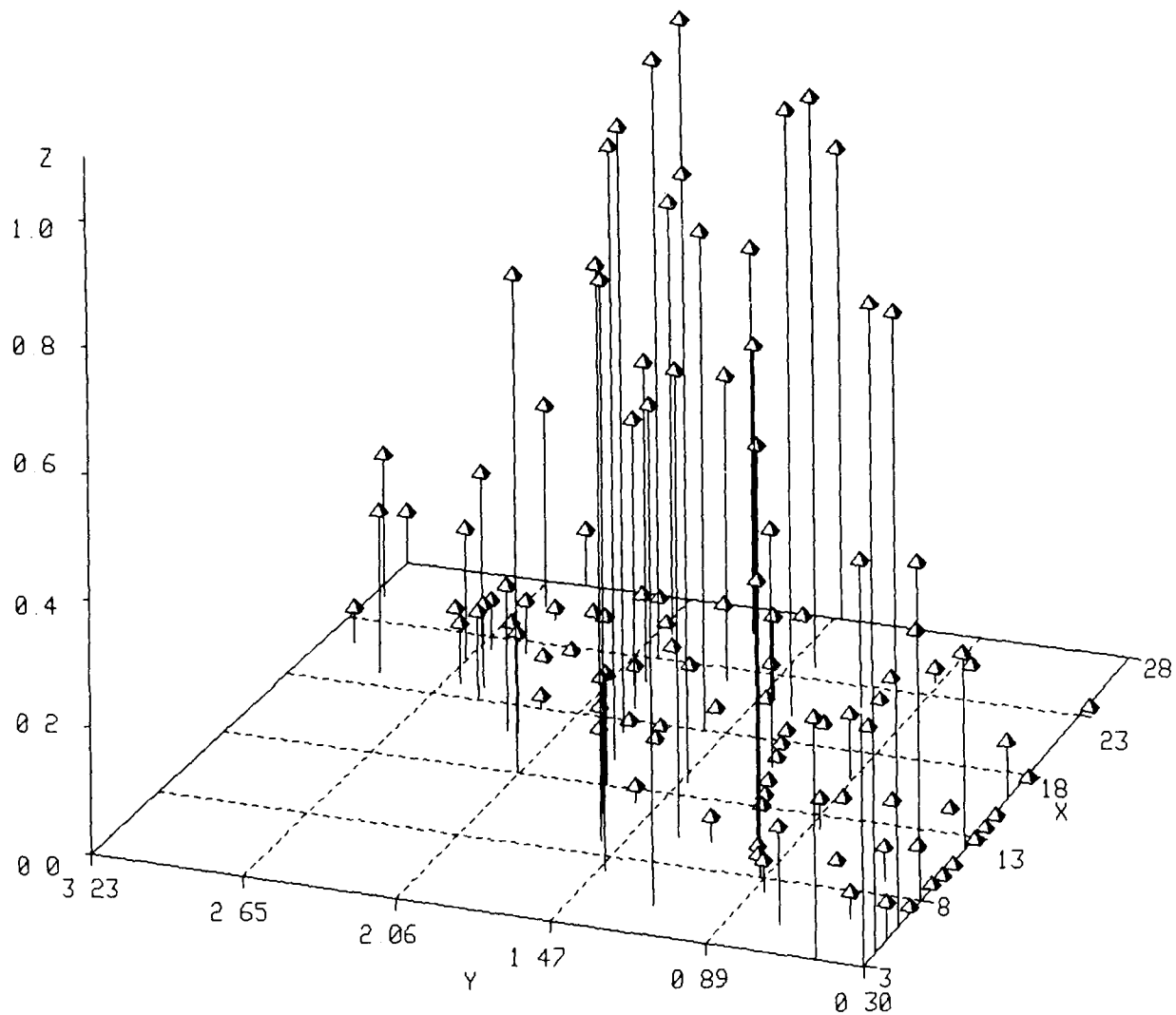


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D28. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 41

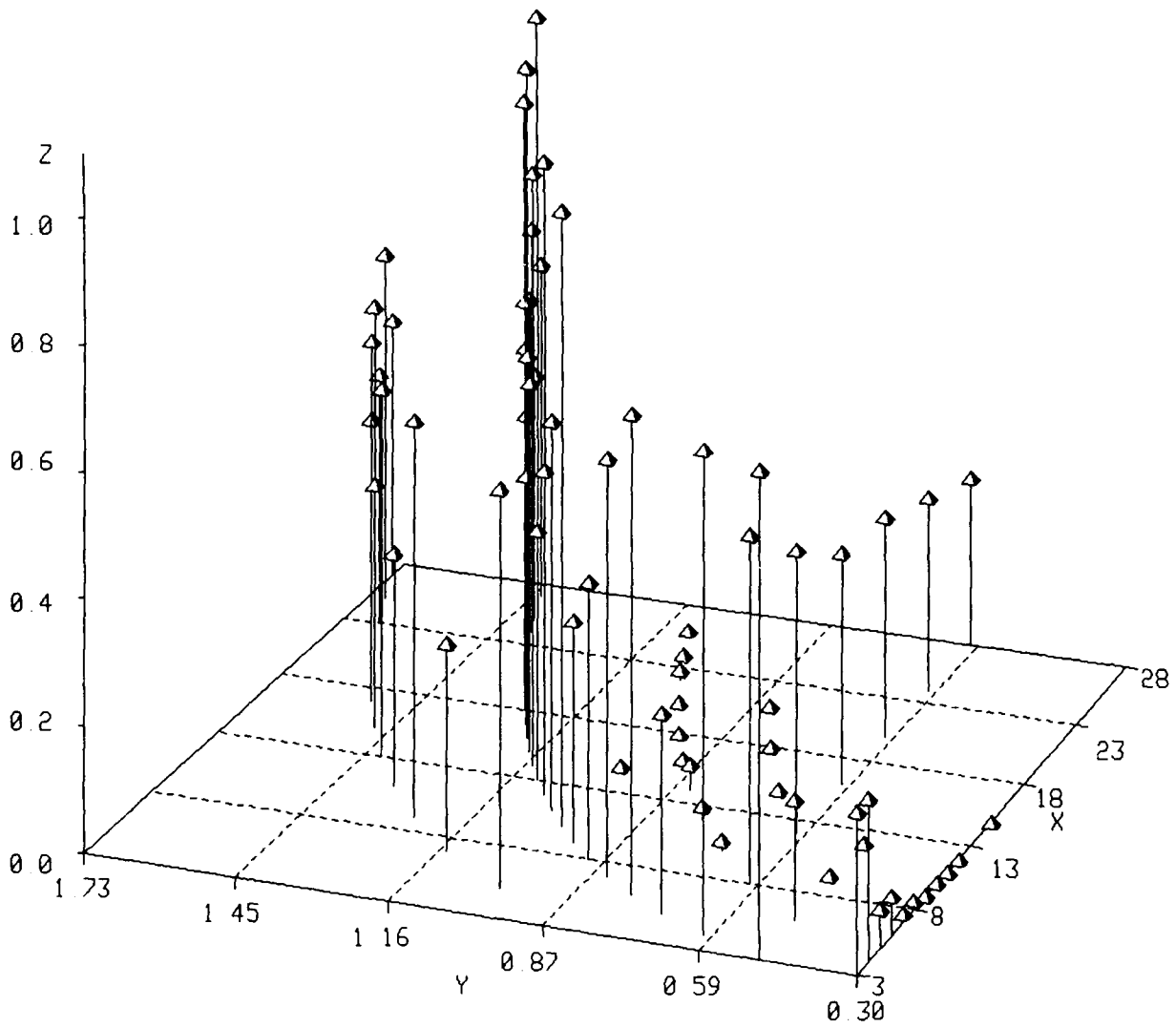


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D29. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 43

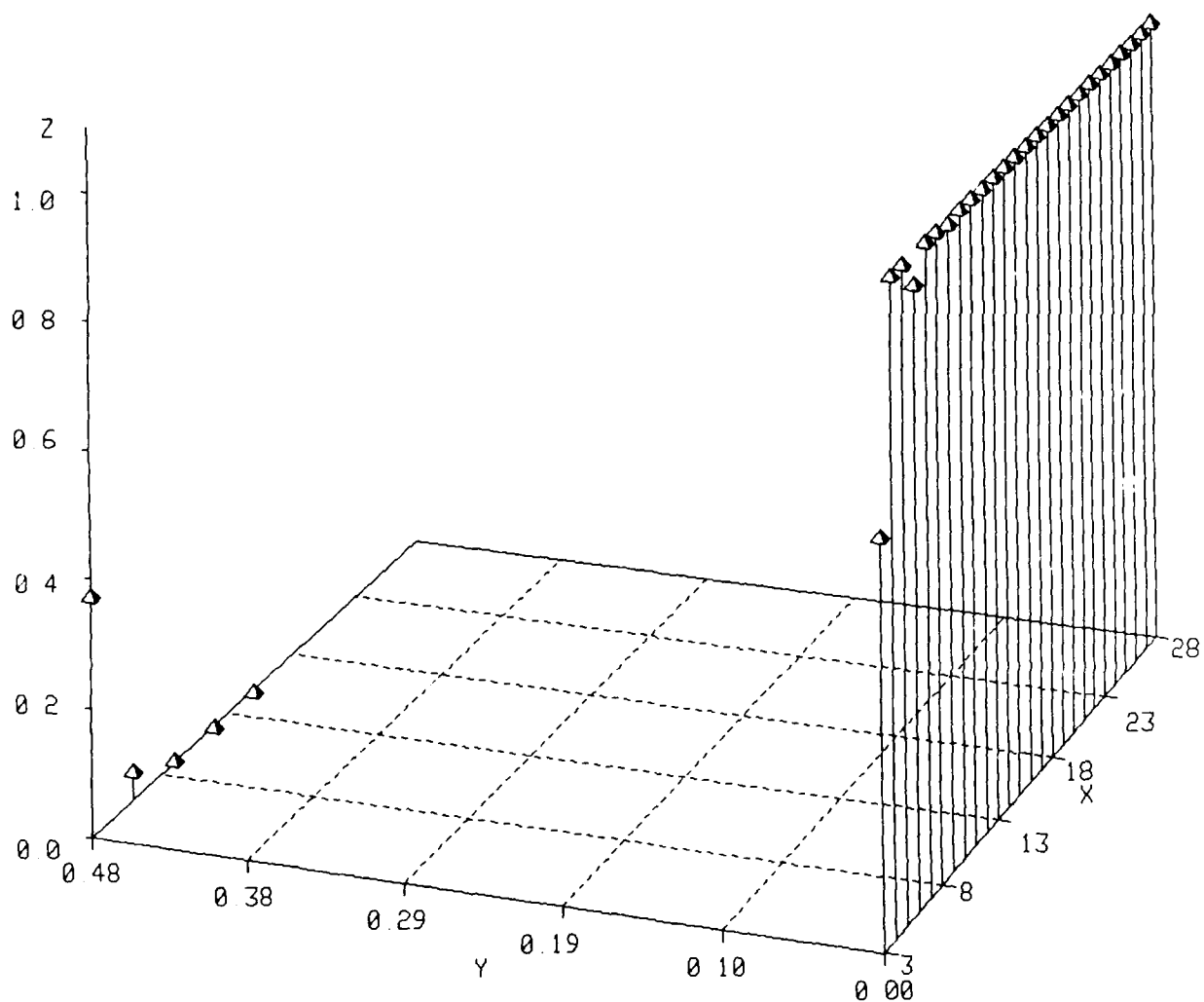


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D30. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 44

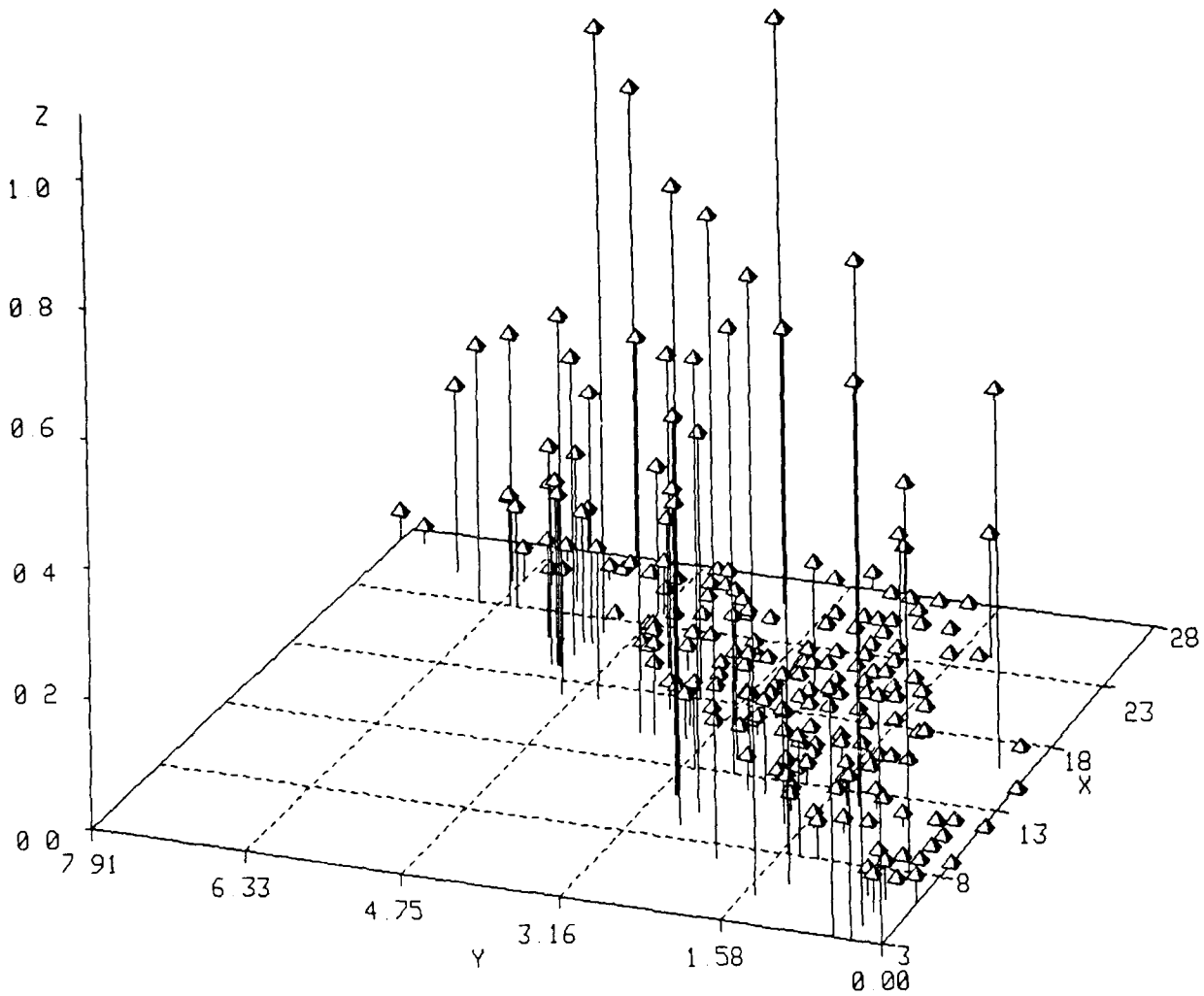


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D31. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 45

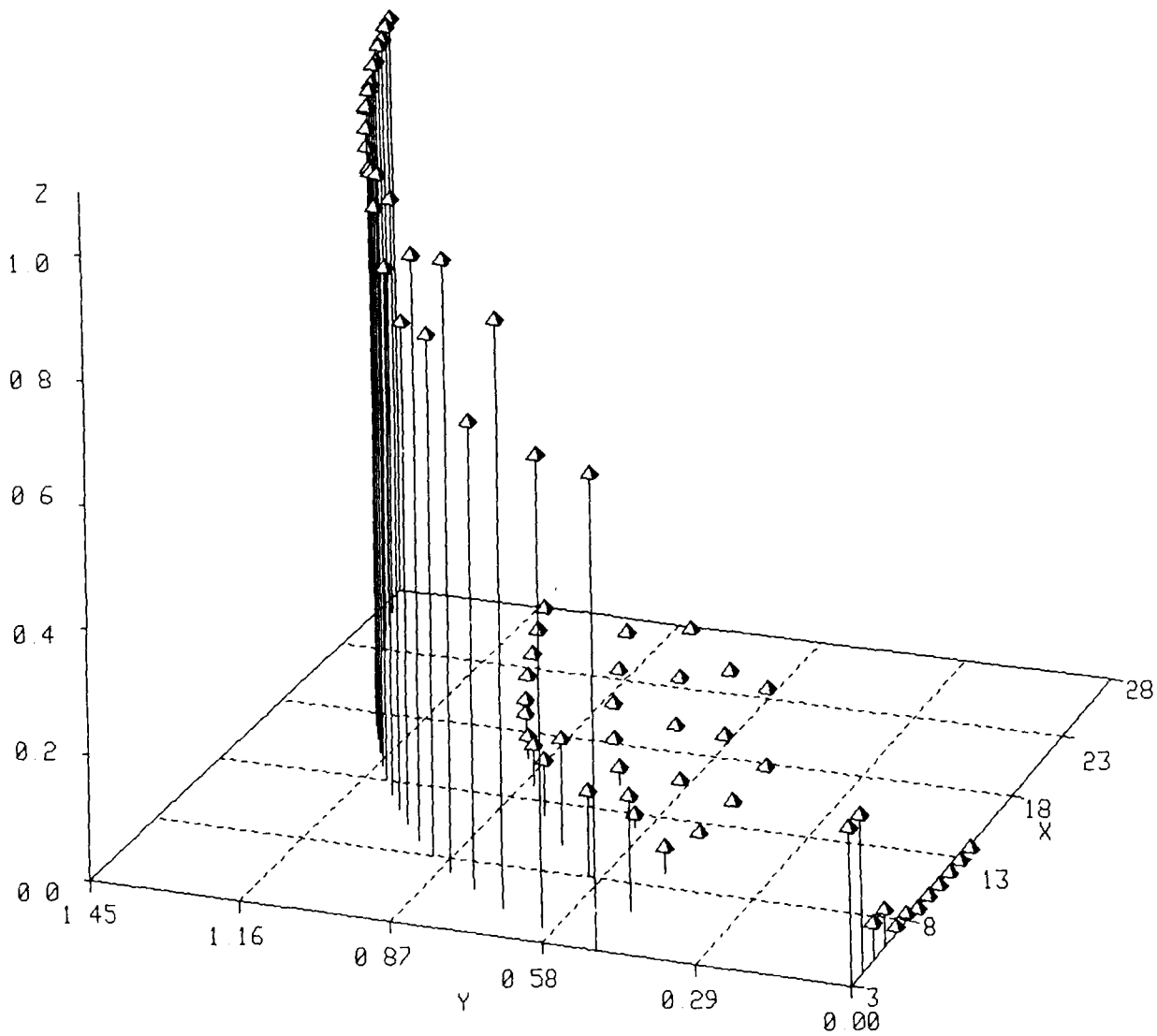


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D32. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 46

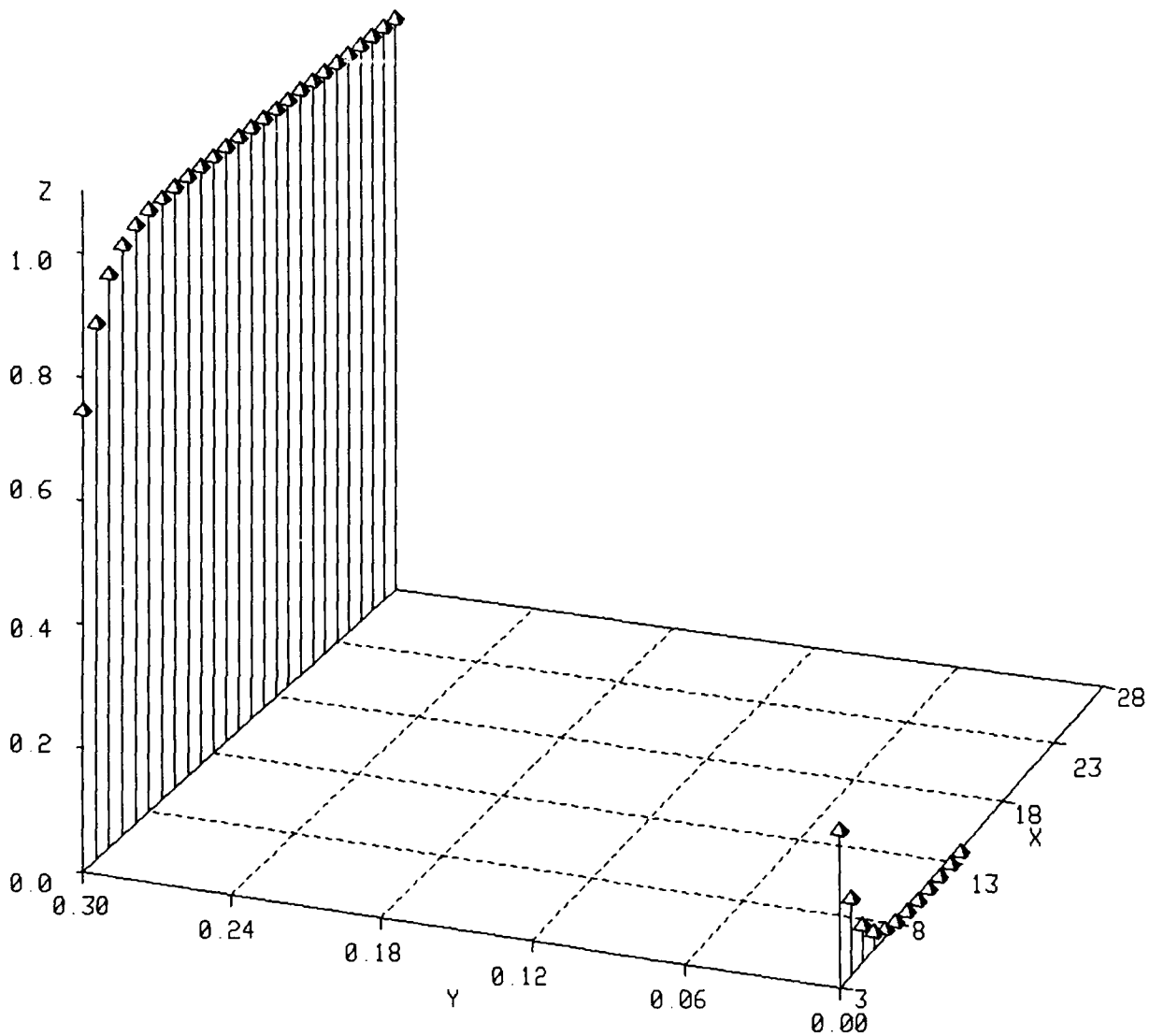


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D33. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 50

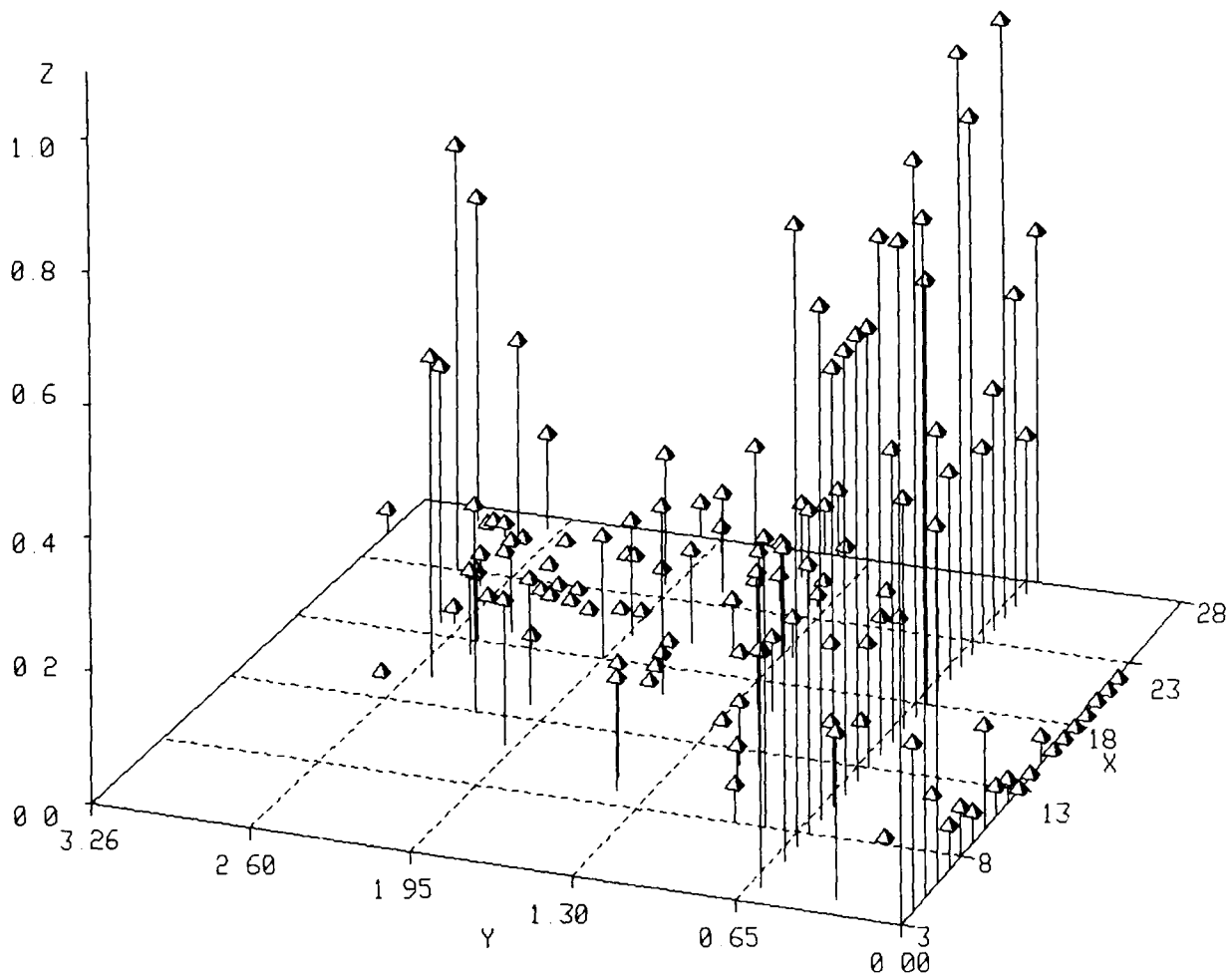


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D34. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 54

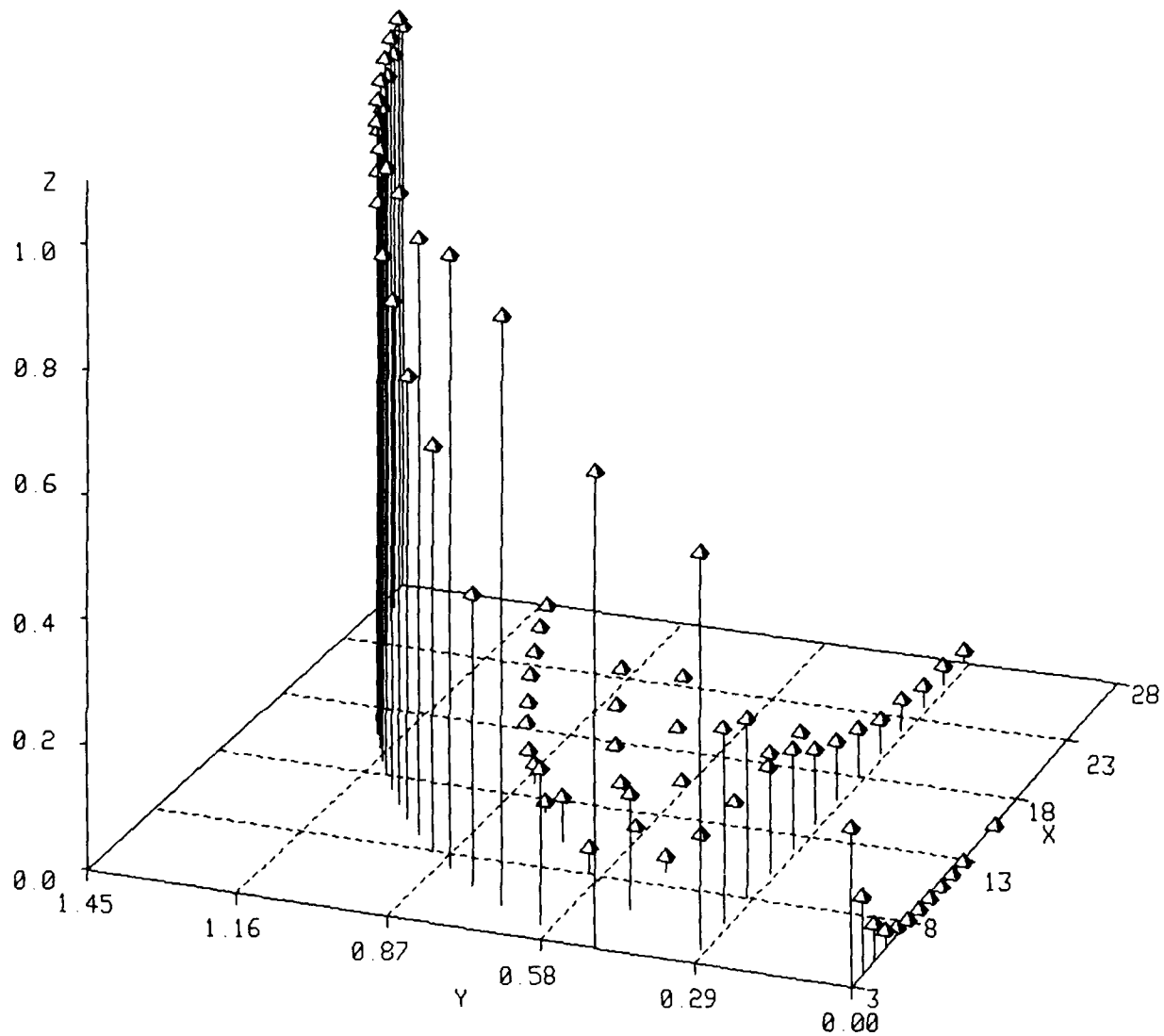


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D35. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 56

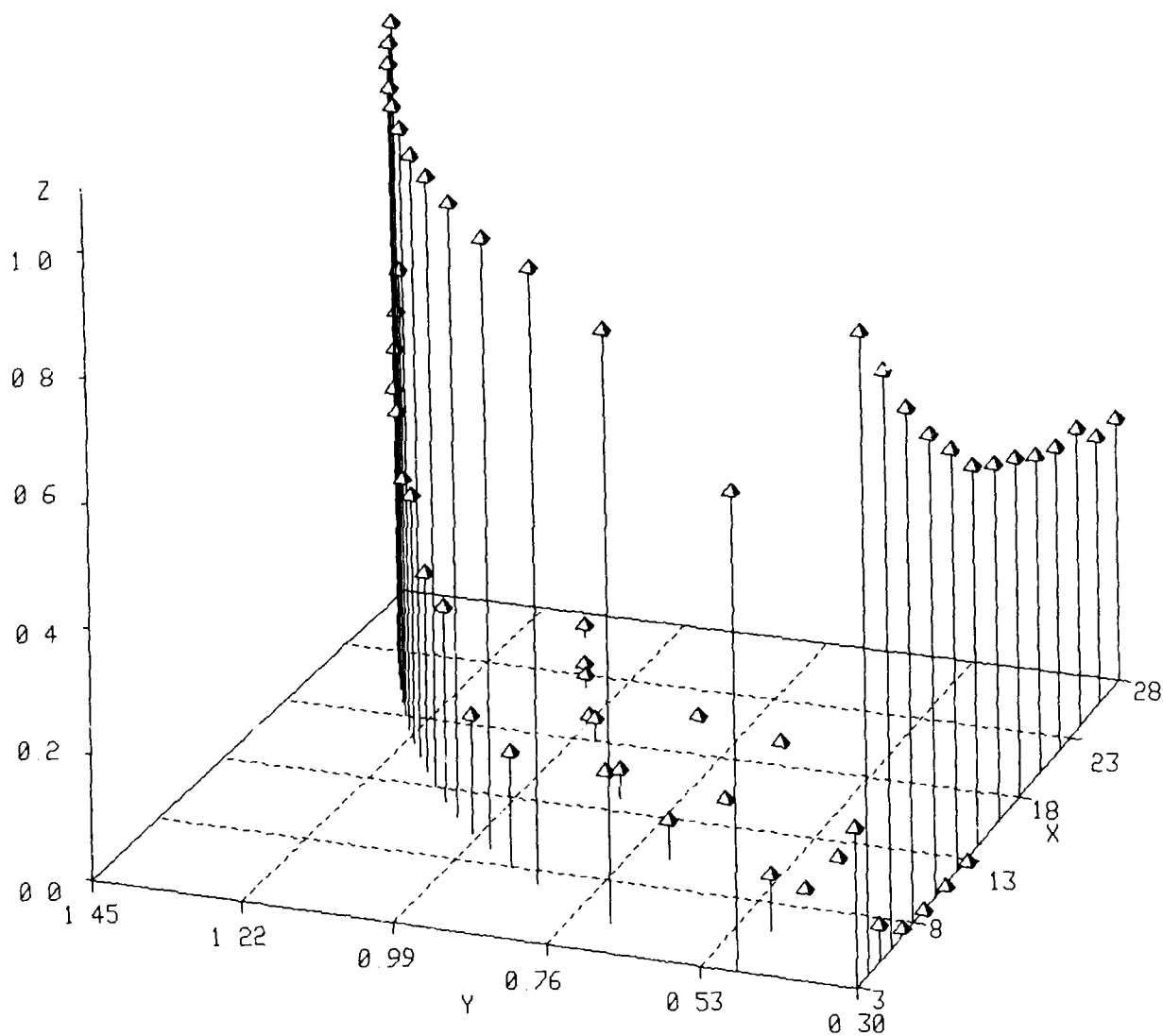


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D36. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 57

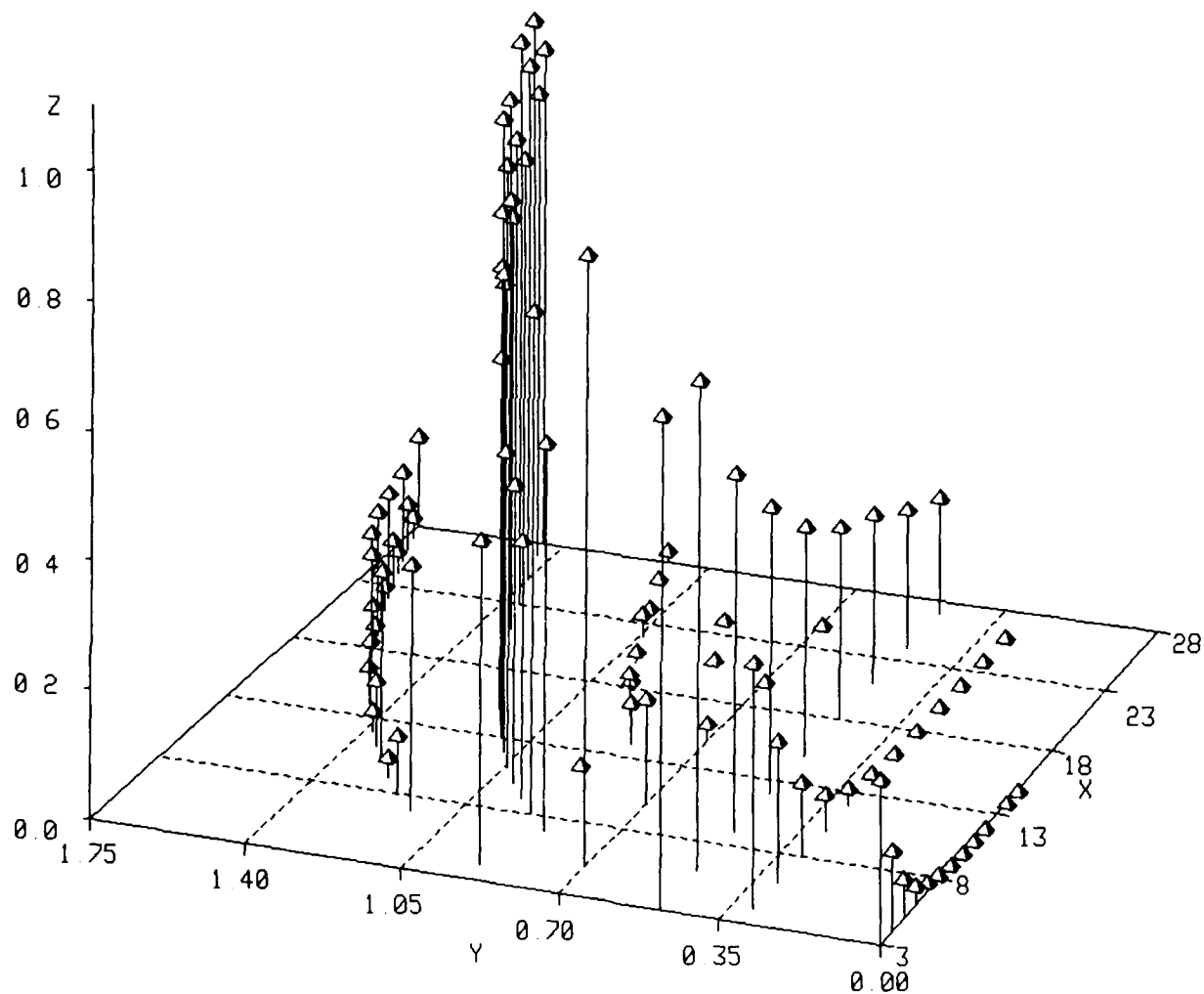


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D37. Attractor probabilities vs array size. Rule is defined in Appendix A.

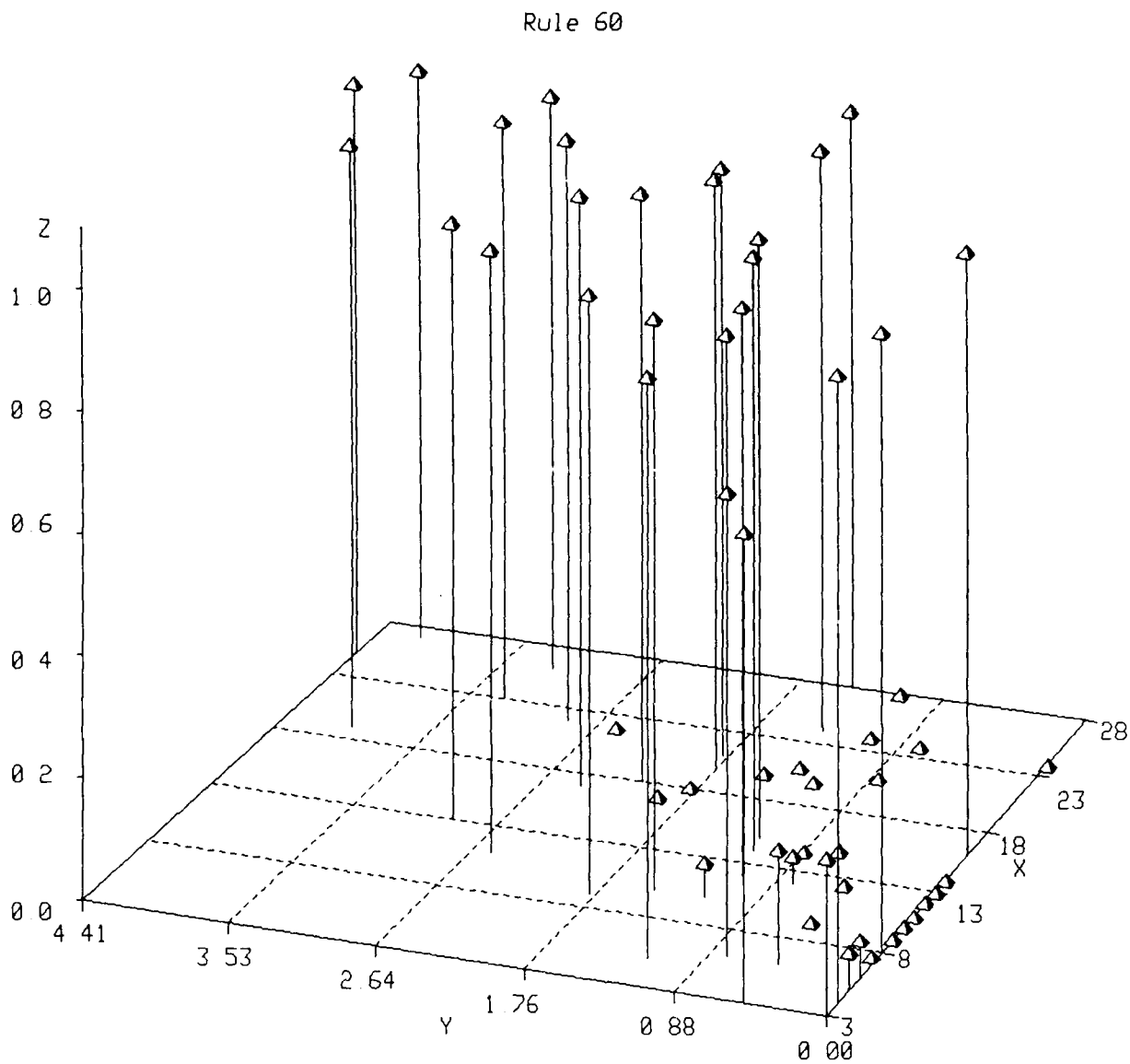
Rule 58



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D38. Attractor probabilities vs array size. Rule is defined in Appendix A.

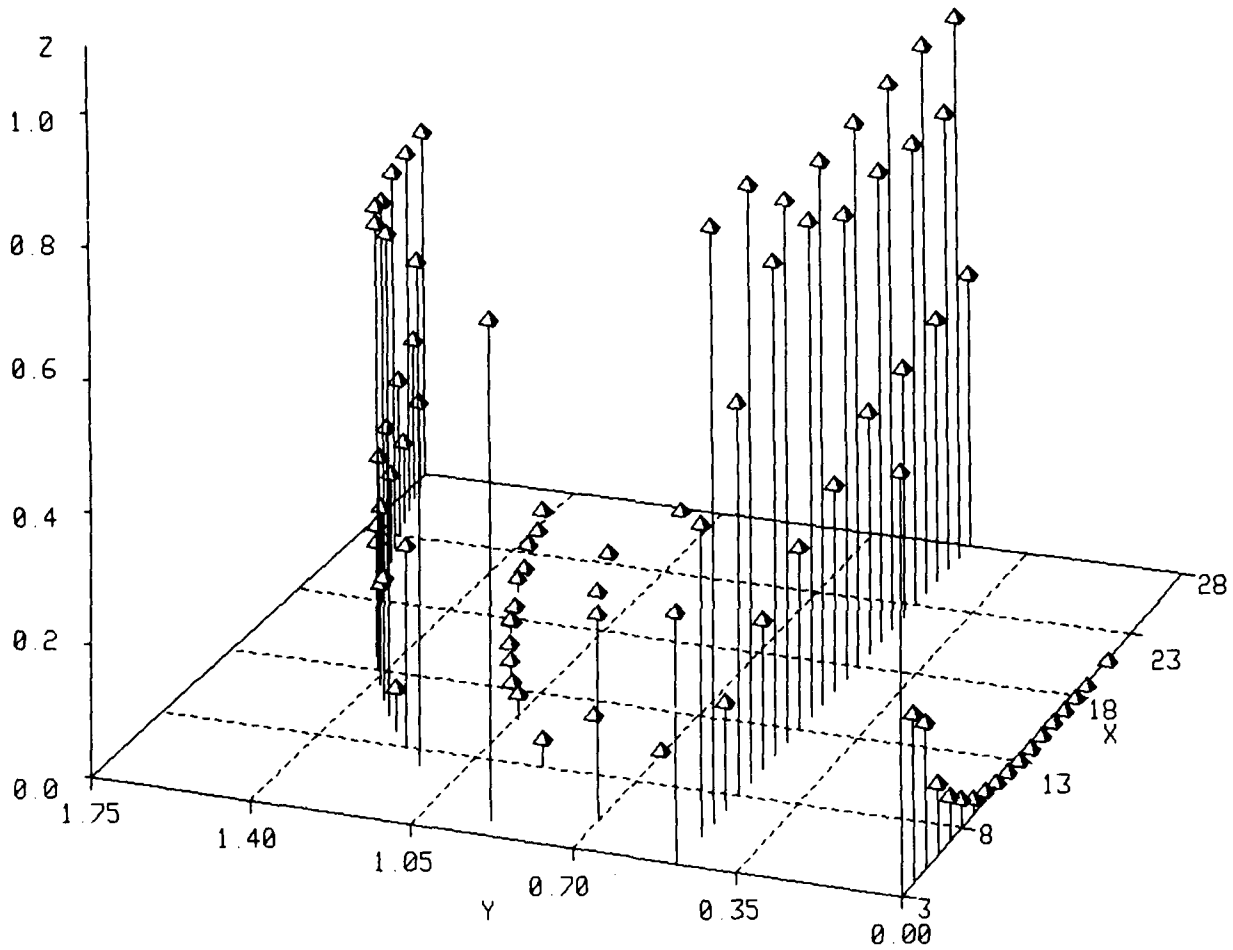


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D39. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 62

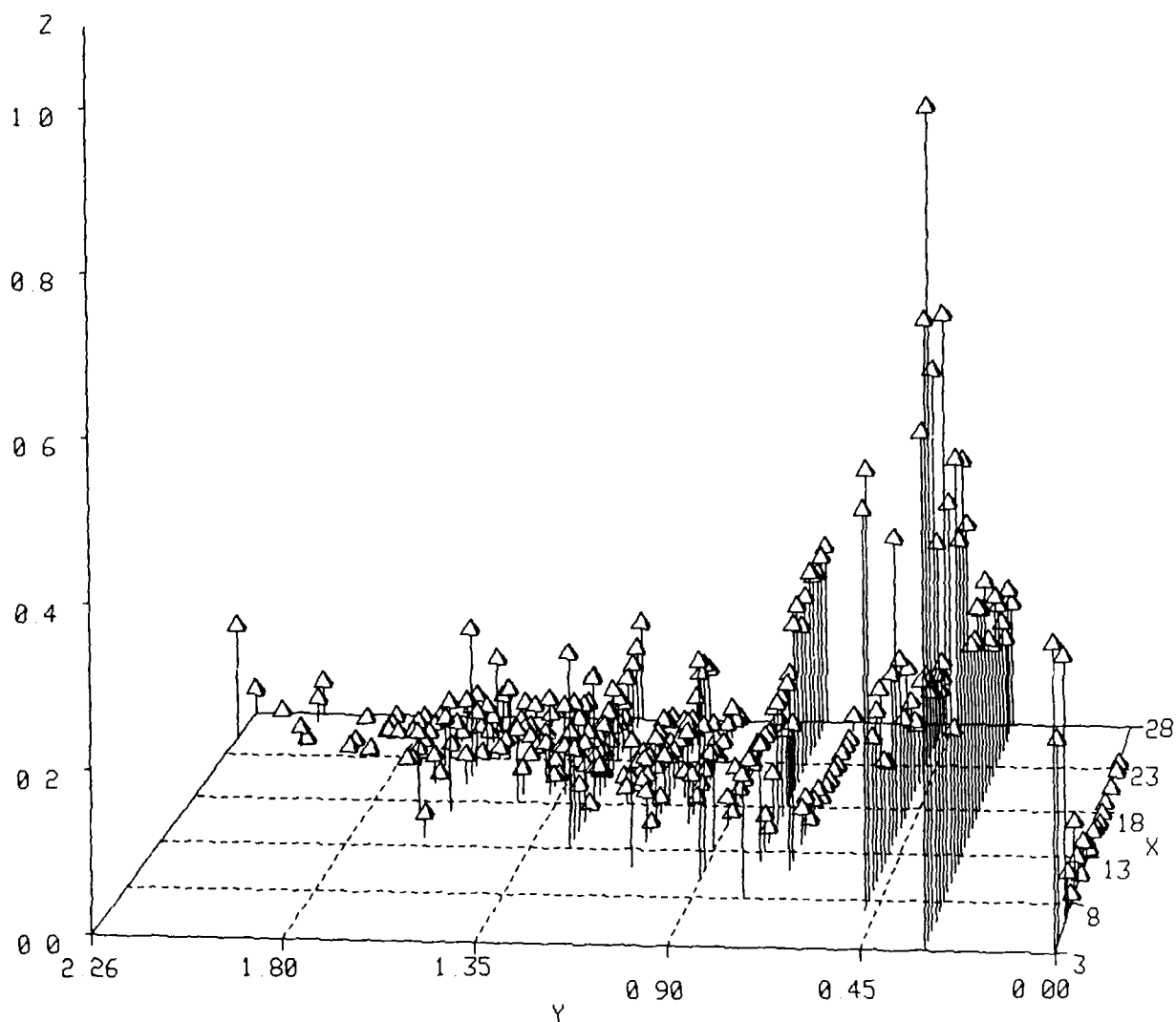


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D40. Attractor probabilities vs array size. Rule is defined in Appendix A.

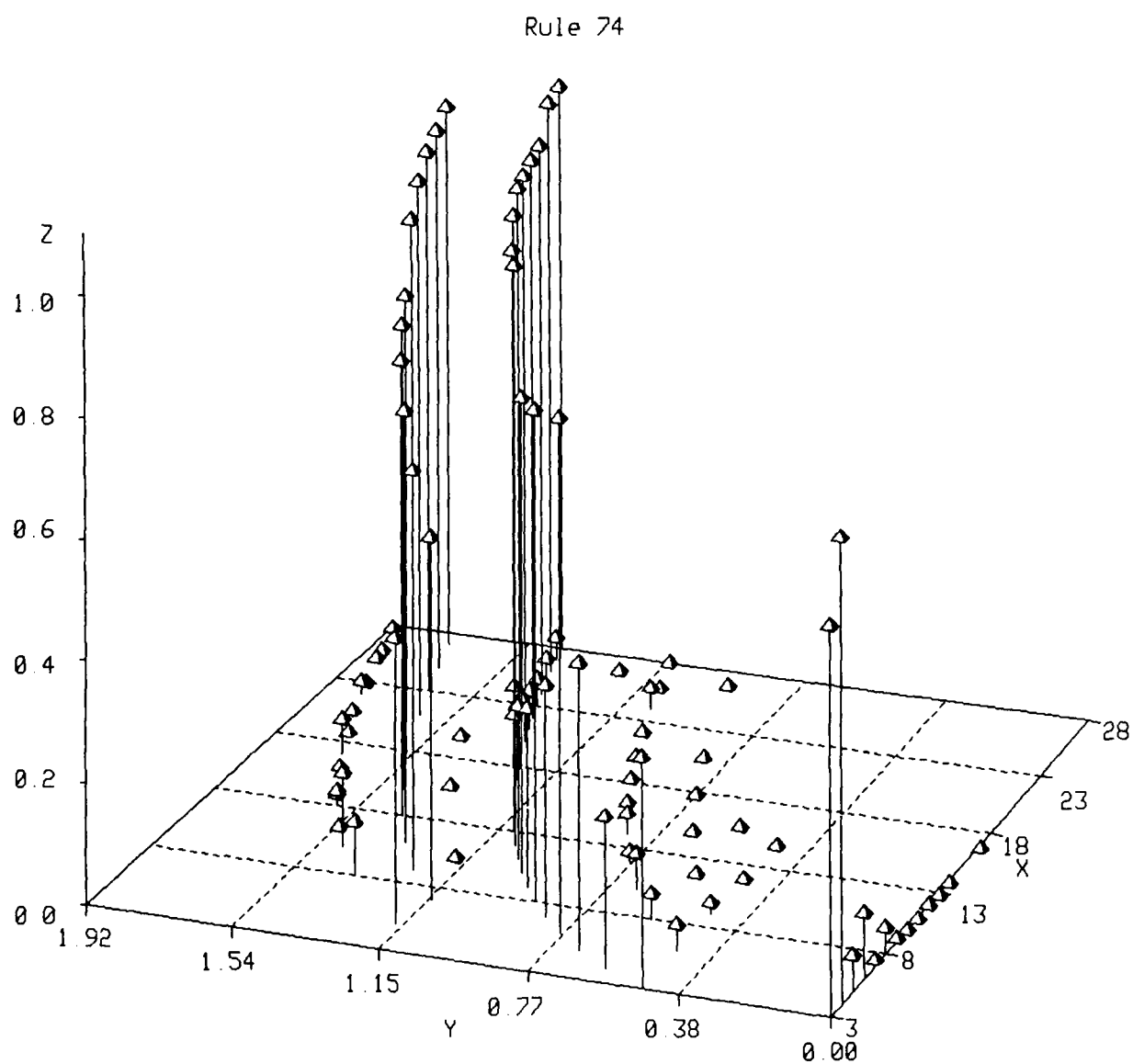
Rule 73



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D41. Attractor probabilities vs array size. Rule is defined in Appendix A.

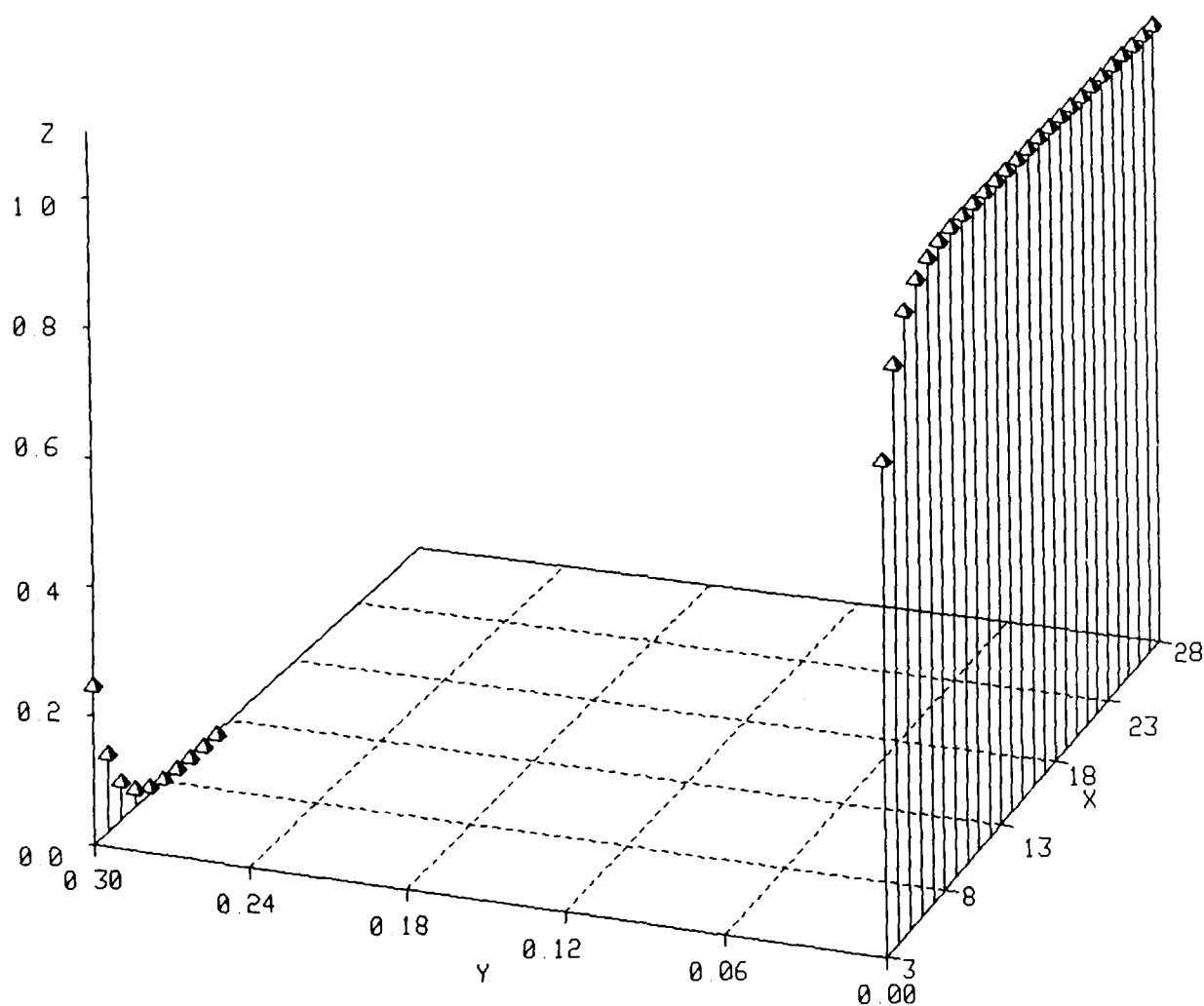


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D42. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 77

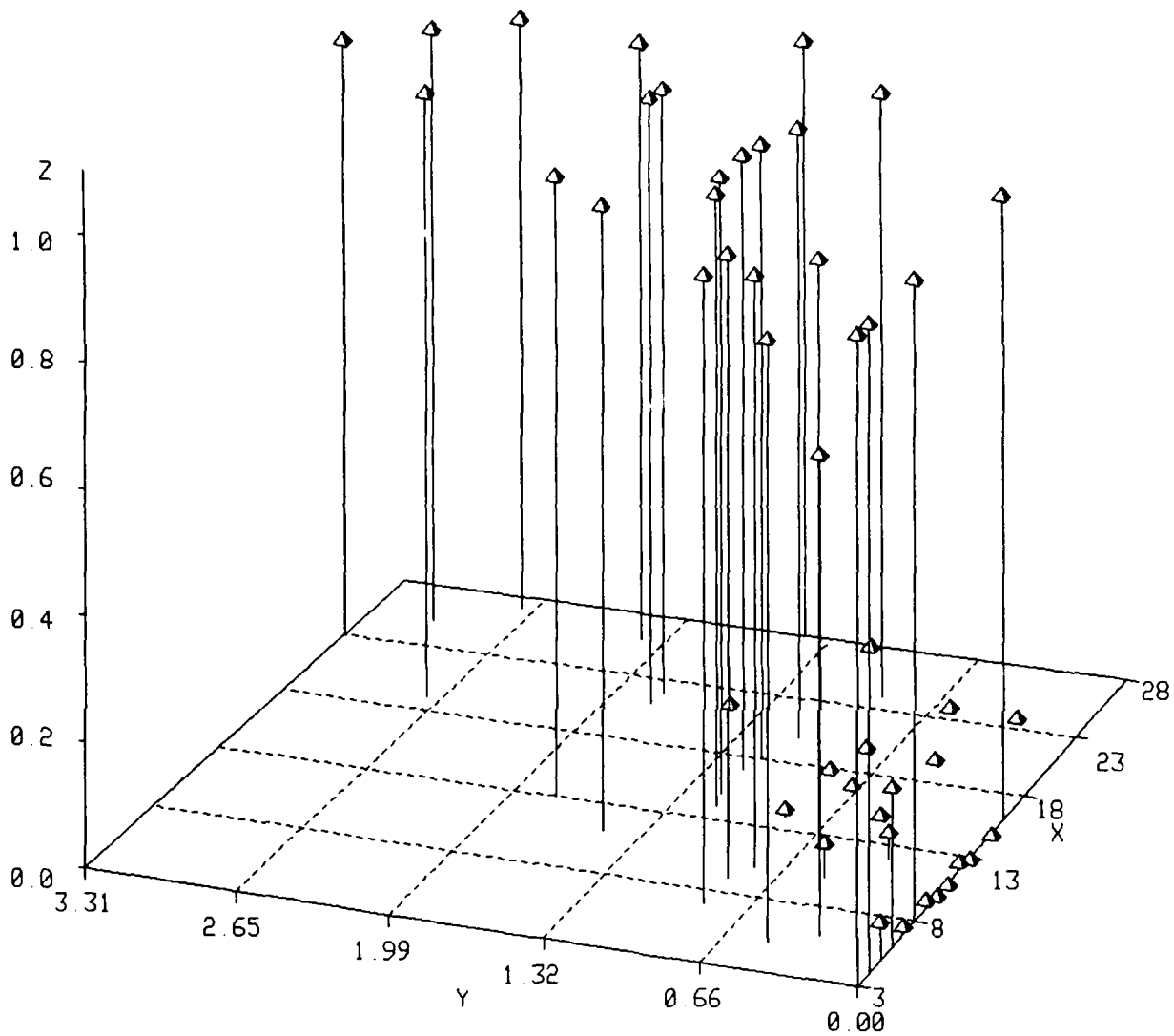


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D43. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 90

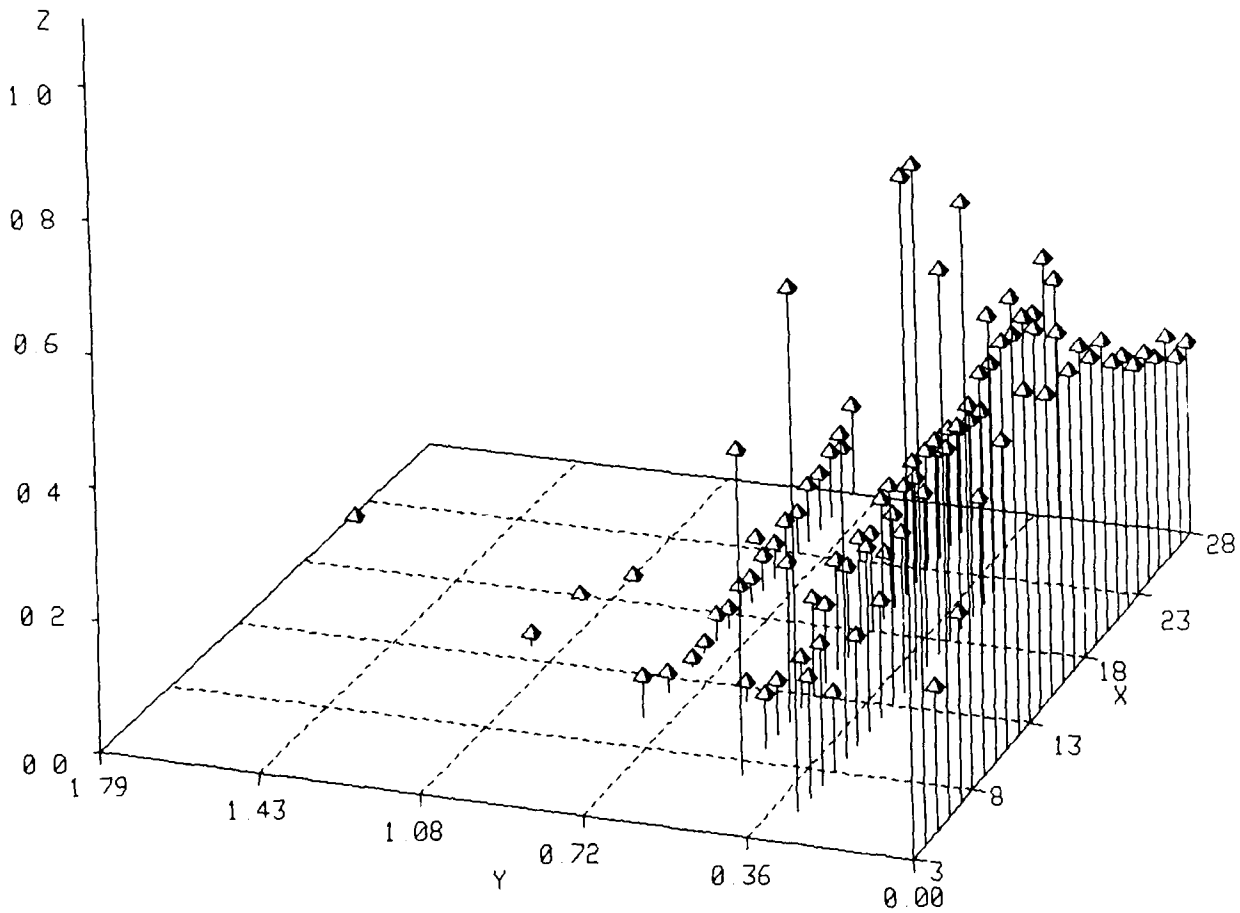


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D44. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 94

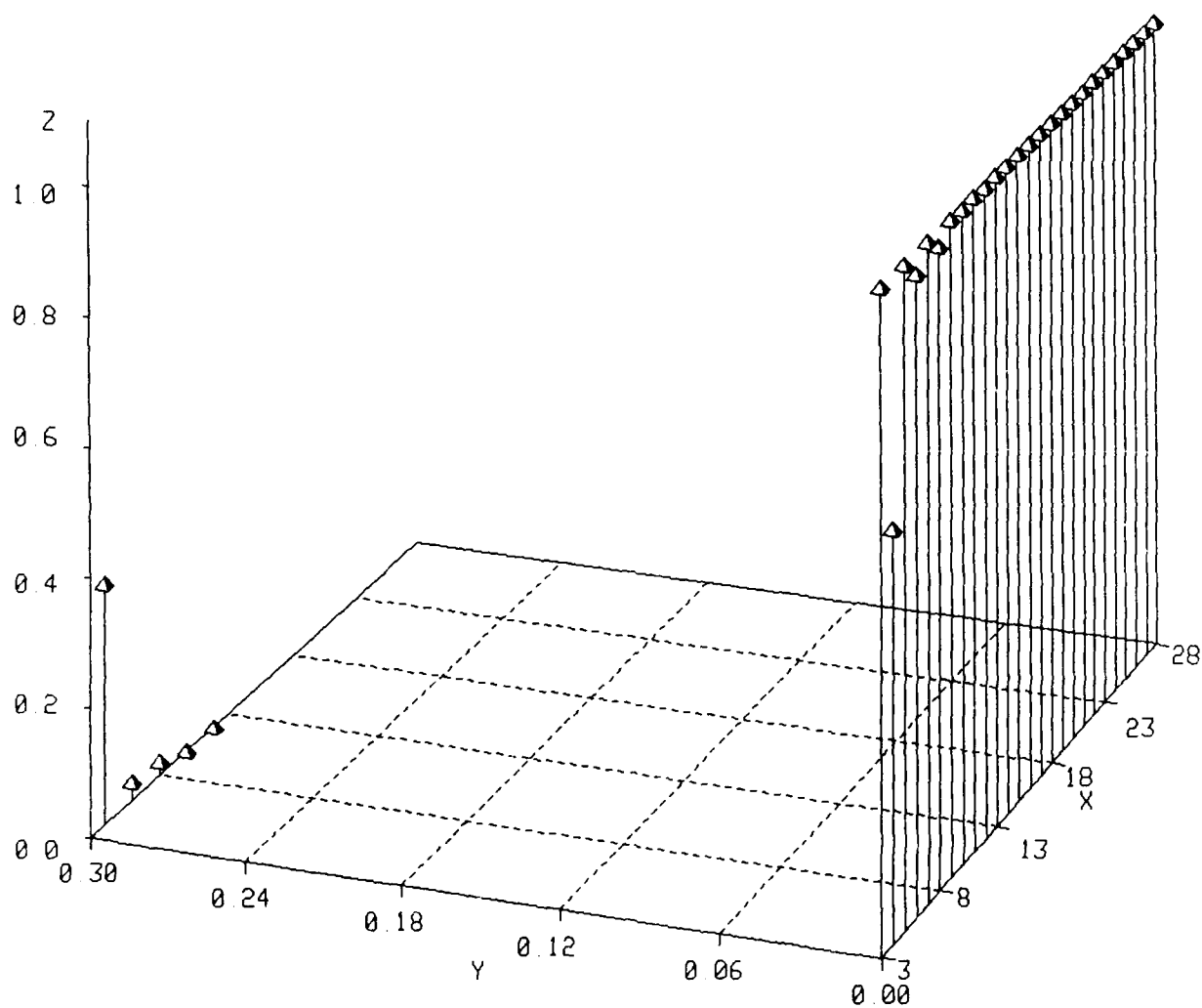


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D45. Attractor probabilities vs array size. Rule is defined in Appendix A.

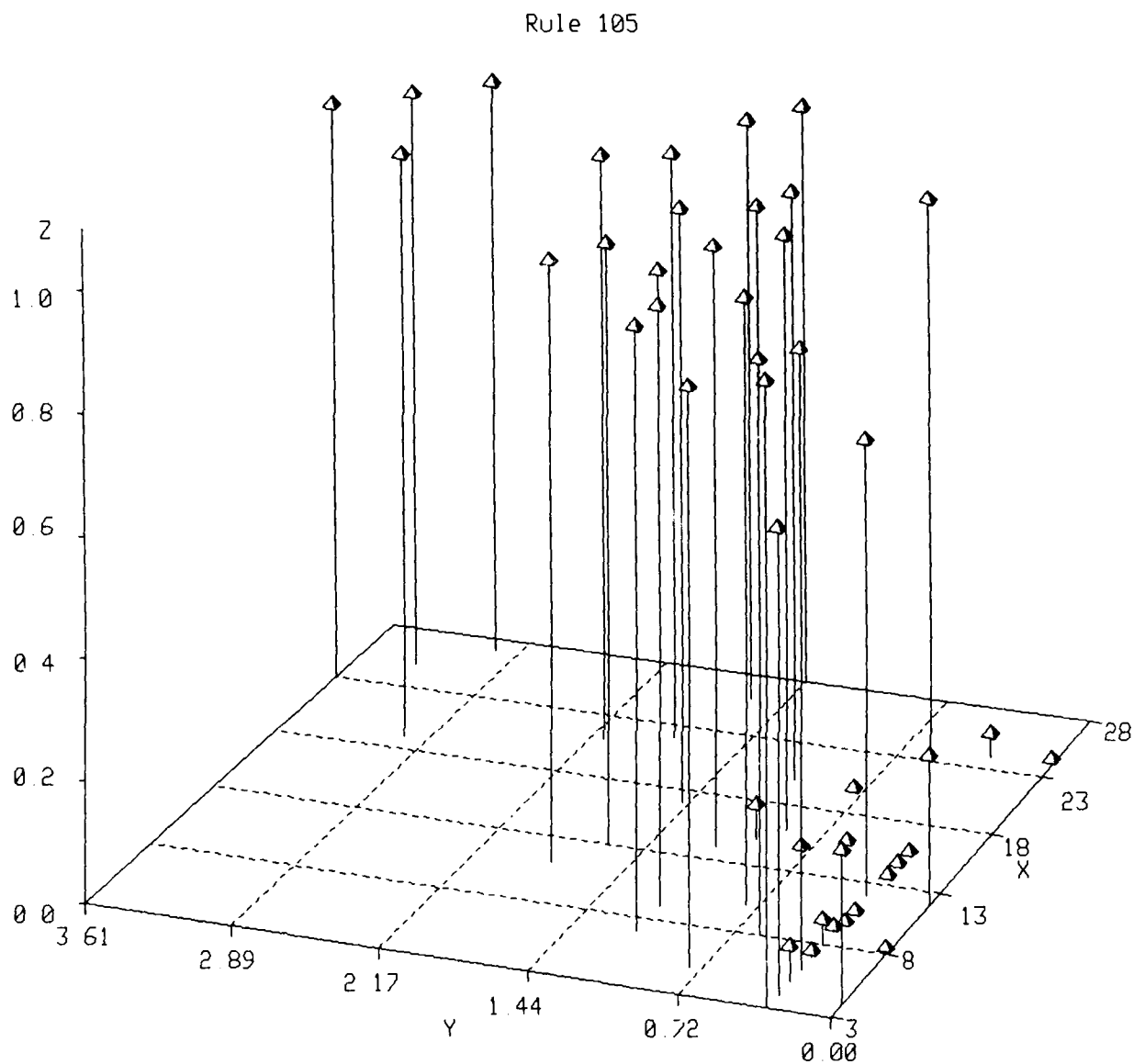
Rule 104



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D46. Attractor probabilities vs array size. Rule is defined in Appendix A.

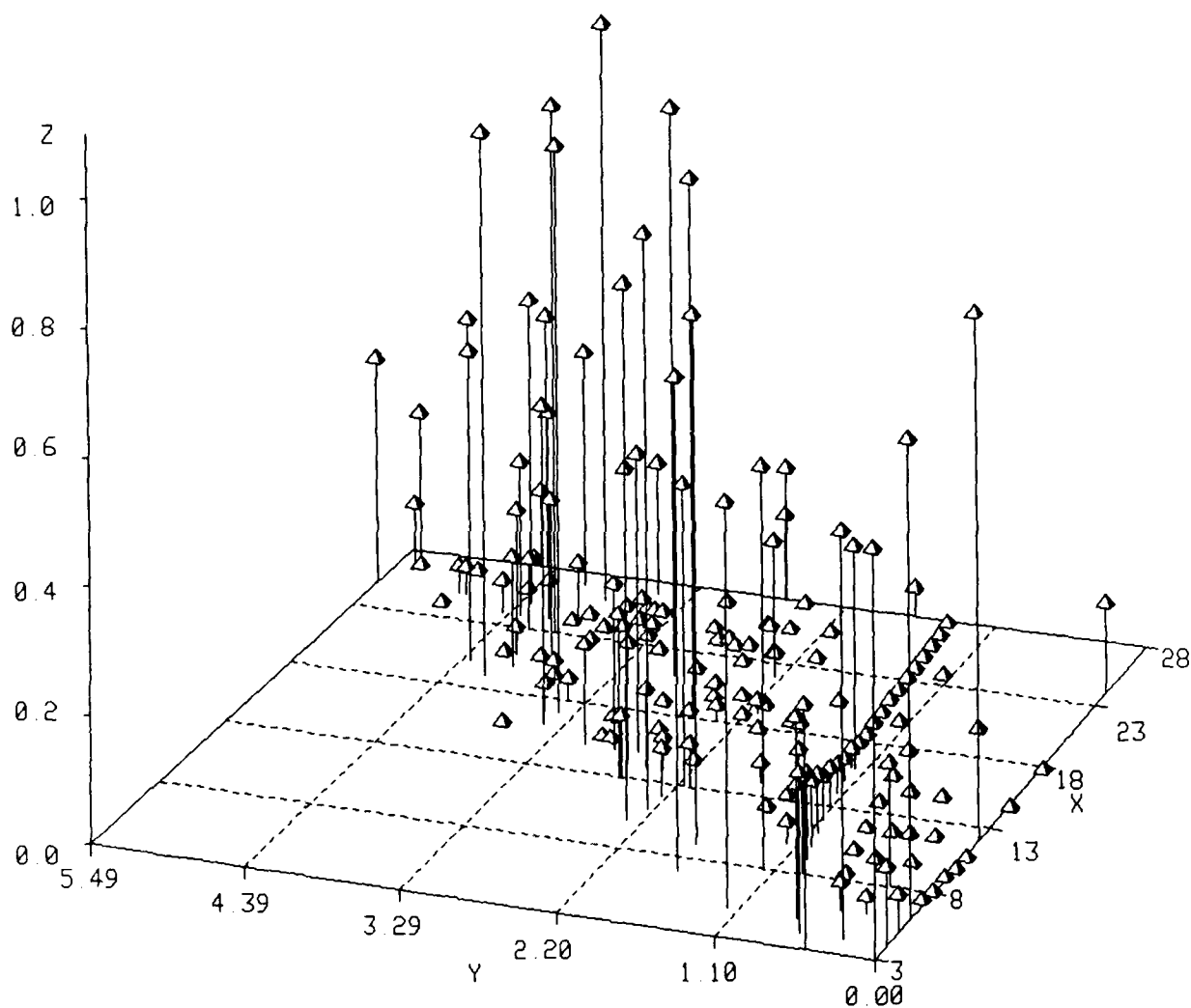


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D47. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 106

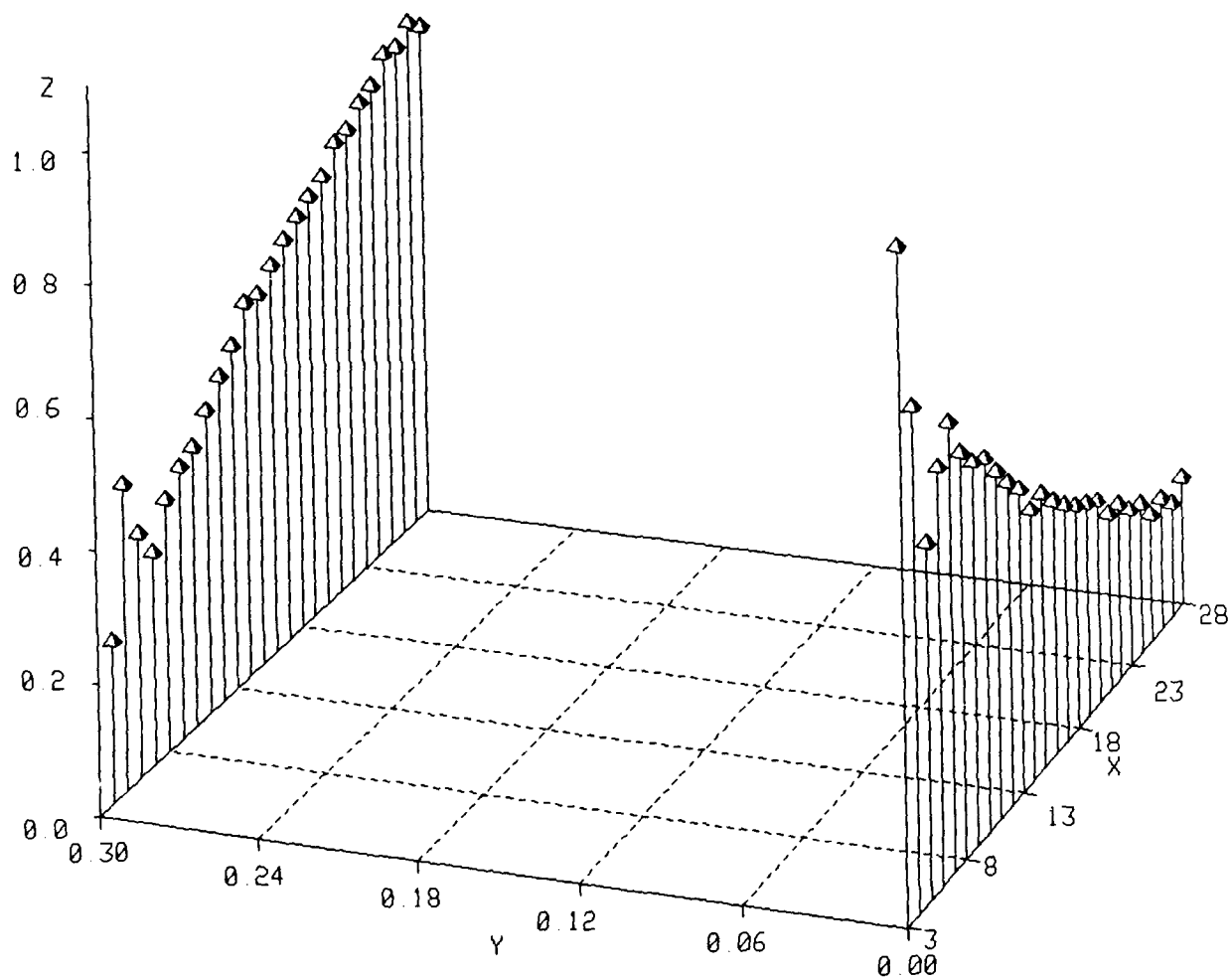


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D48. Attractor probabilities vs array size. Rule is defined in Appendix A.

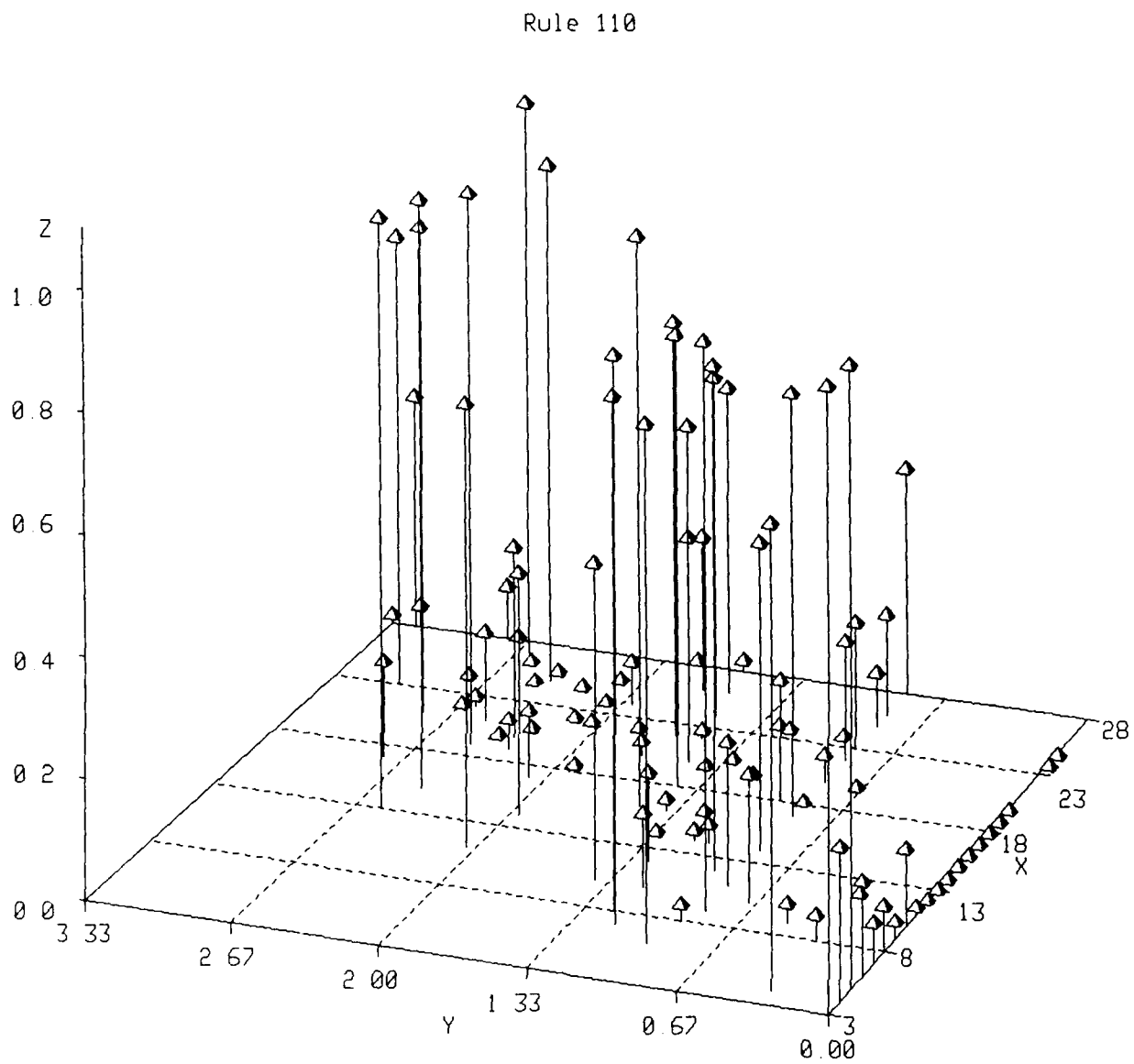
Rule 108



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D49. Attractor probabilities vs array size. Rule is defined in Appendix A.

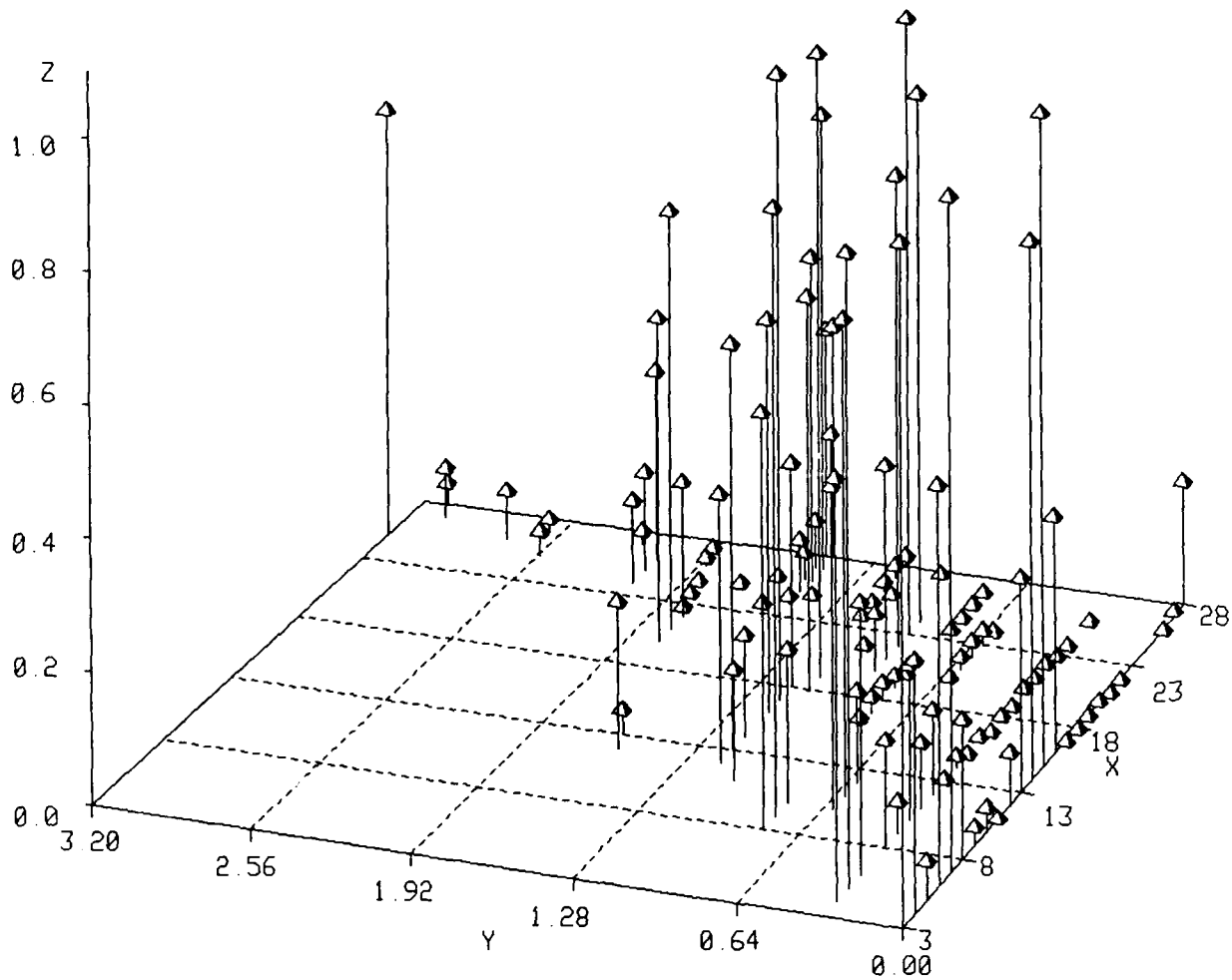


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D50. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 122

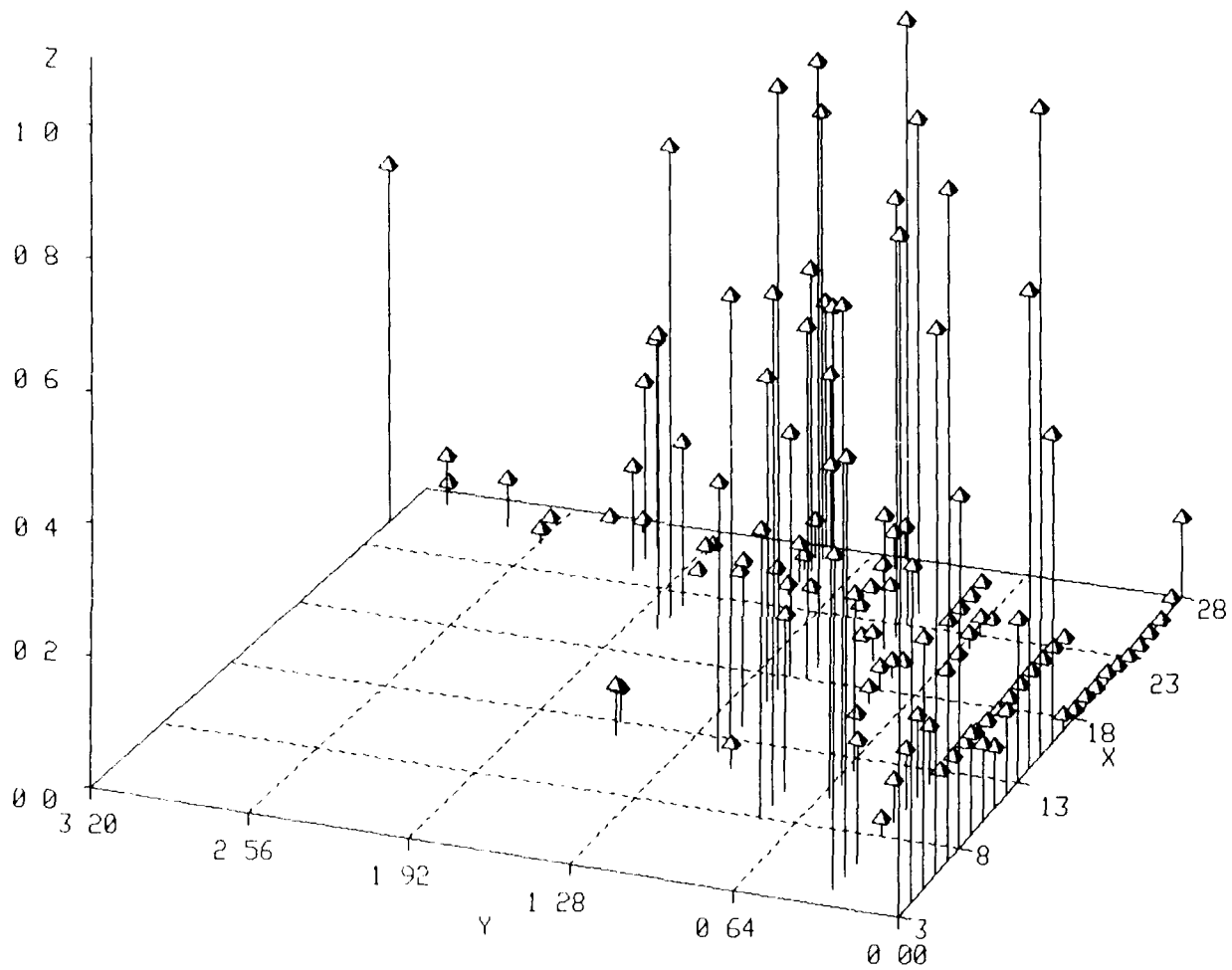


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D51. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 126

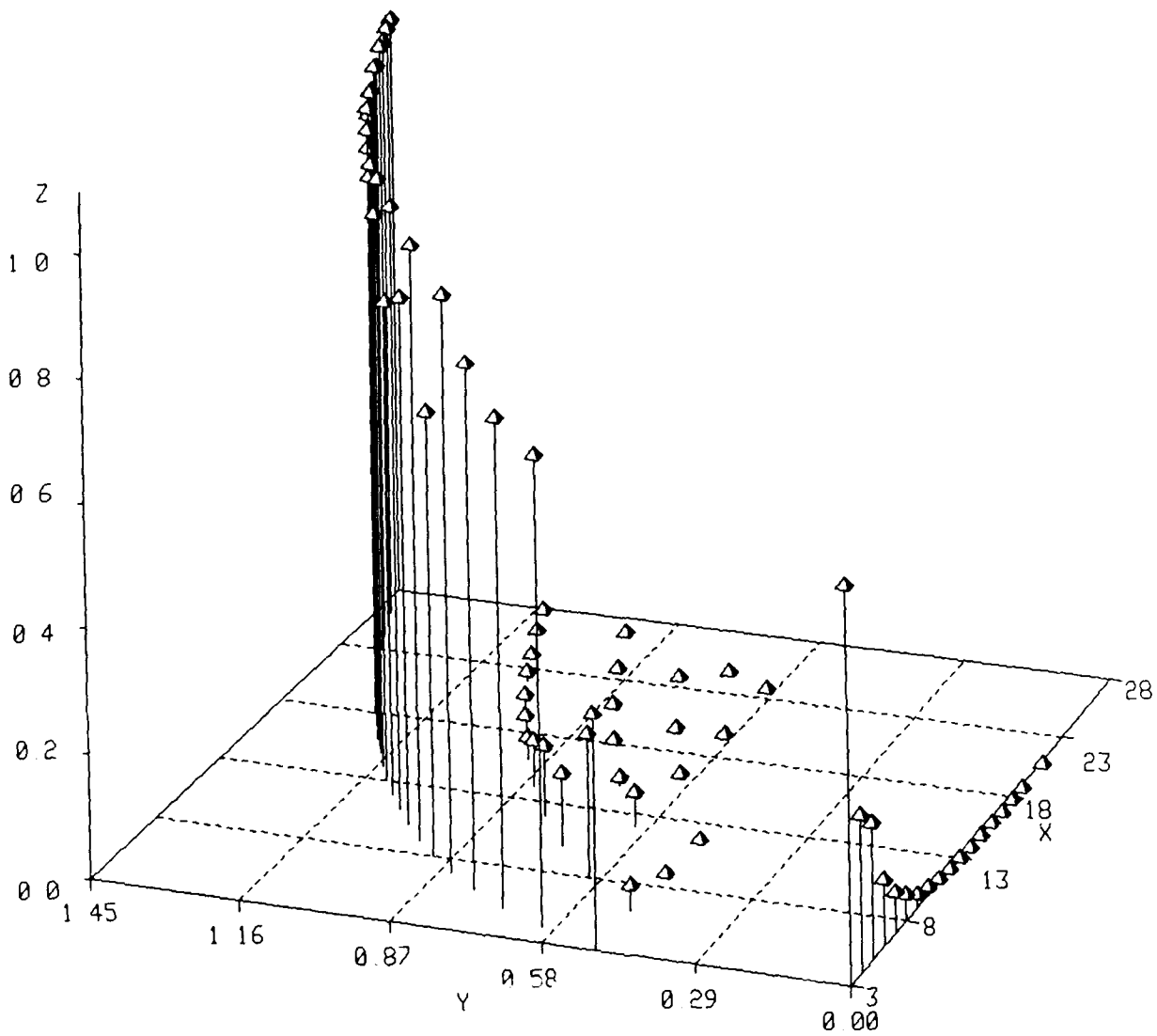


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D52. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 130

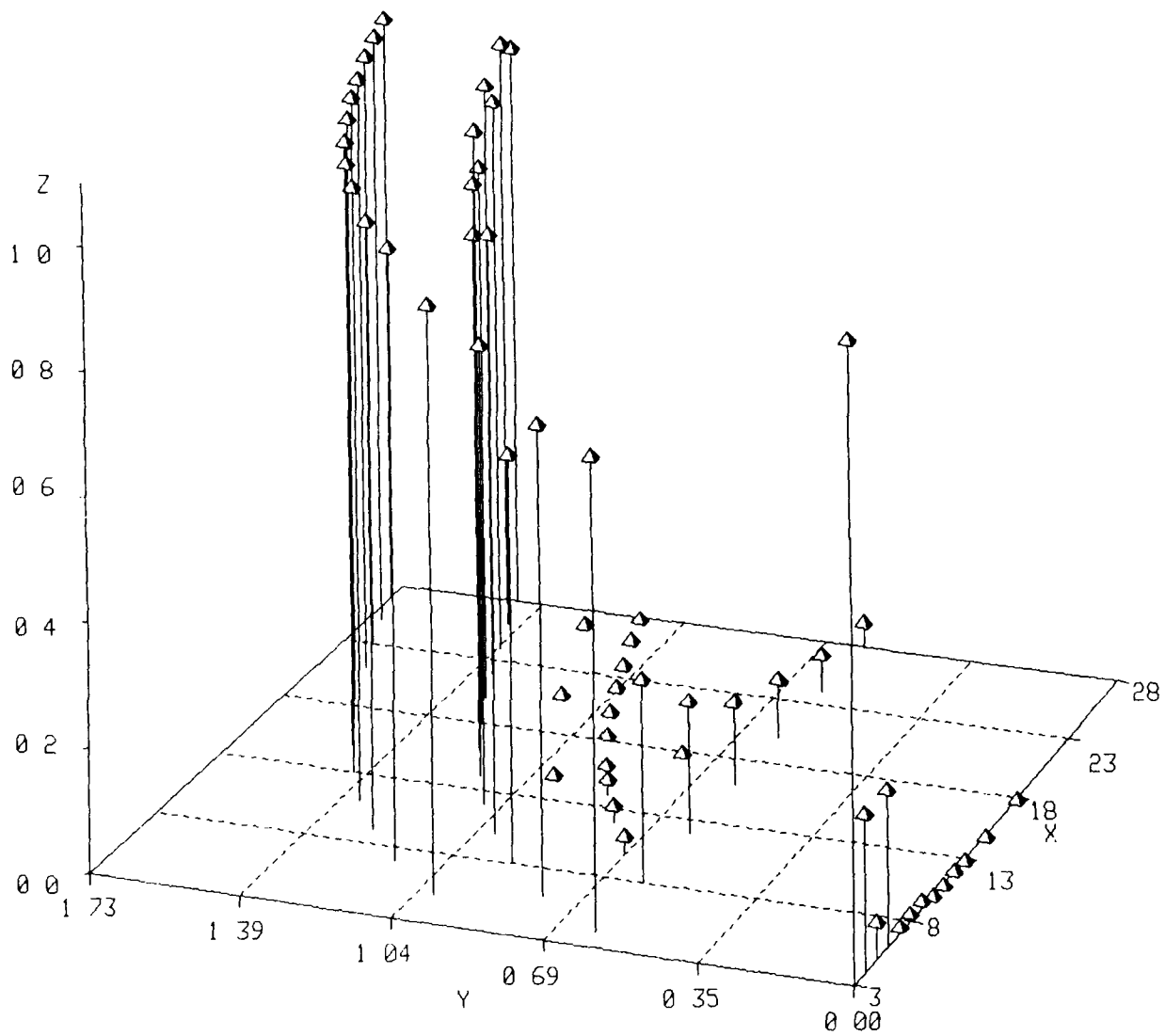


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D53. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 134

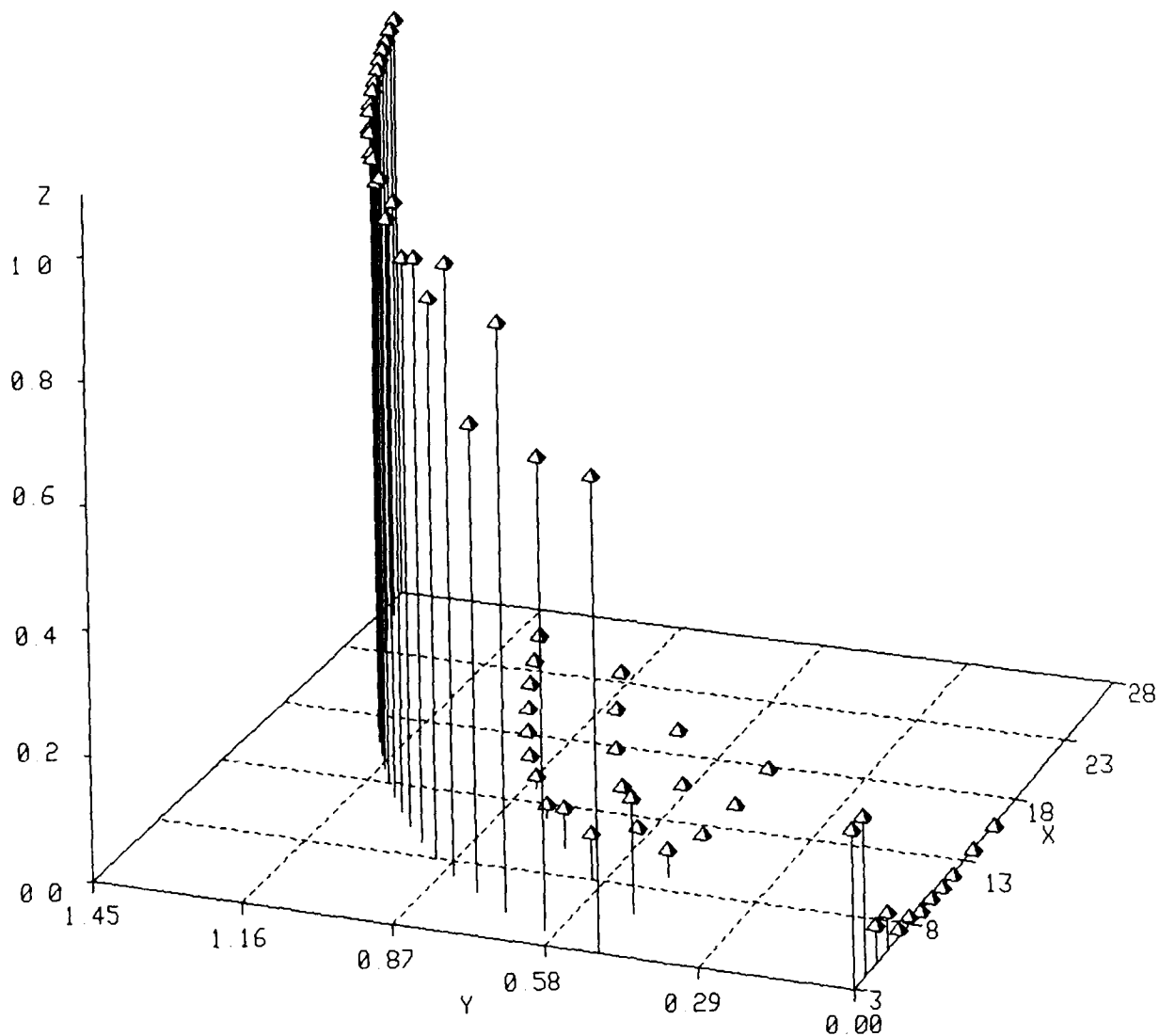


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D54. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 138

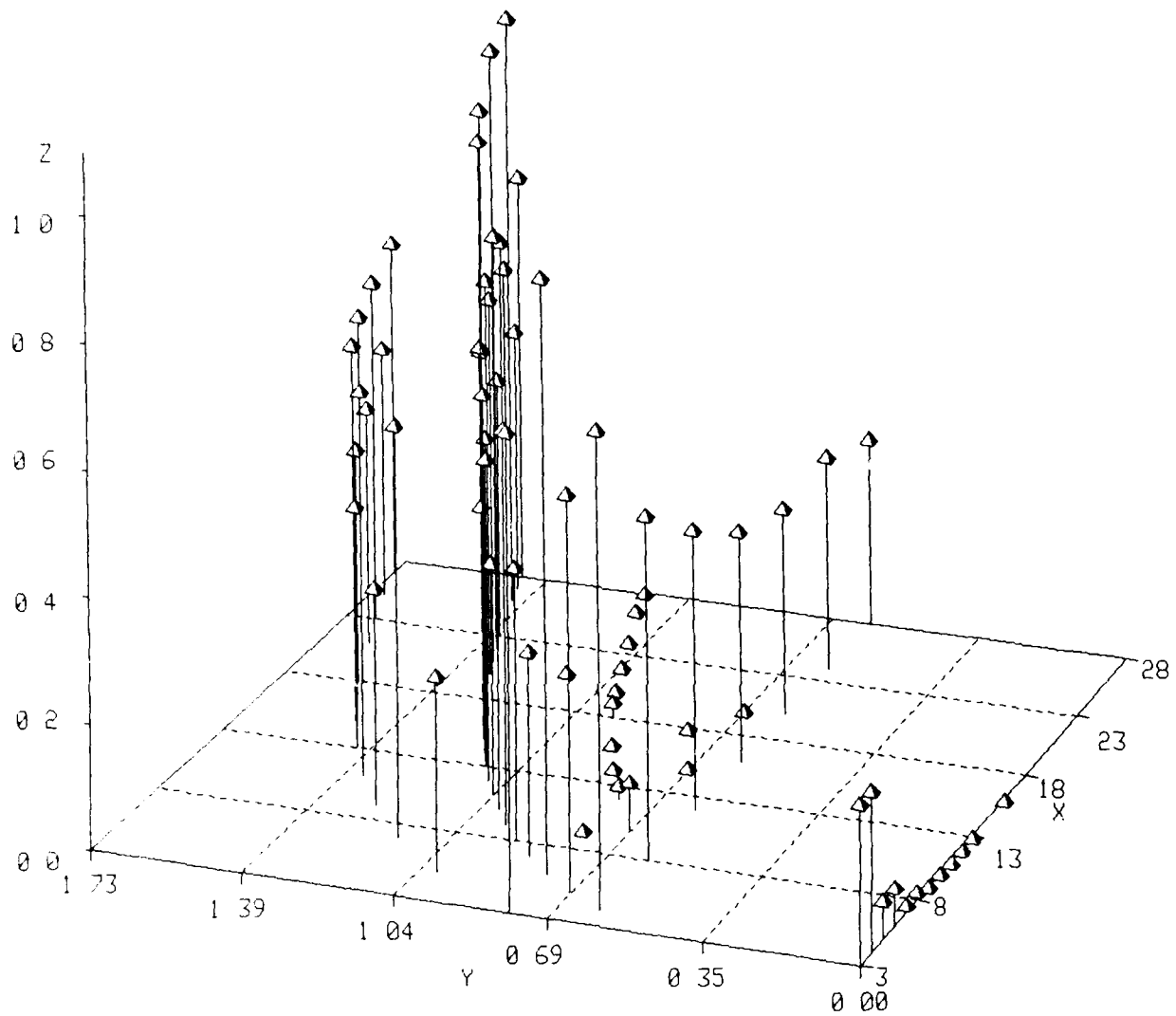


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D55. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 142

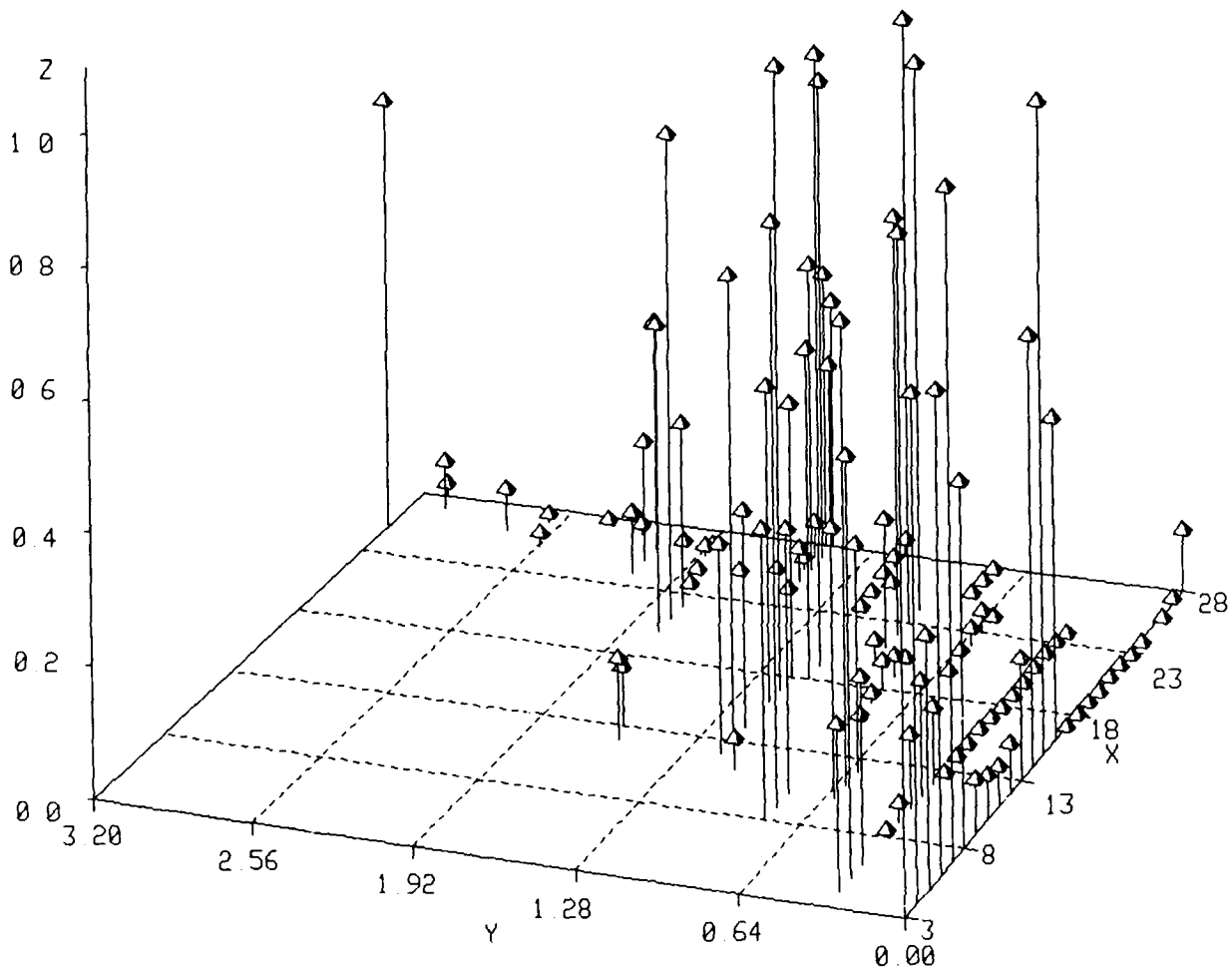


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D56. Attractor probabilities vs array size. Rule is defined in Appendix A.

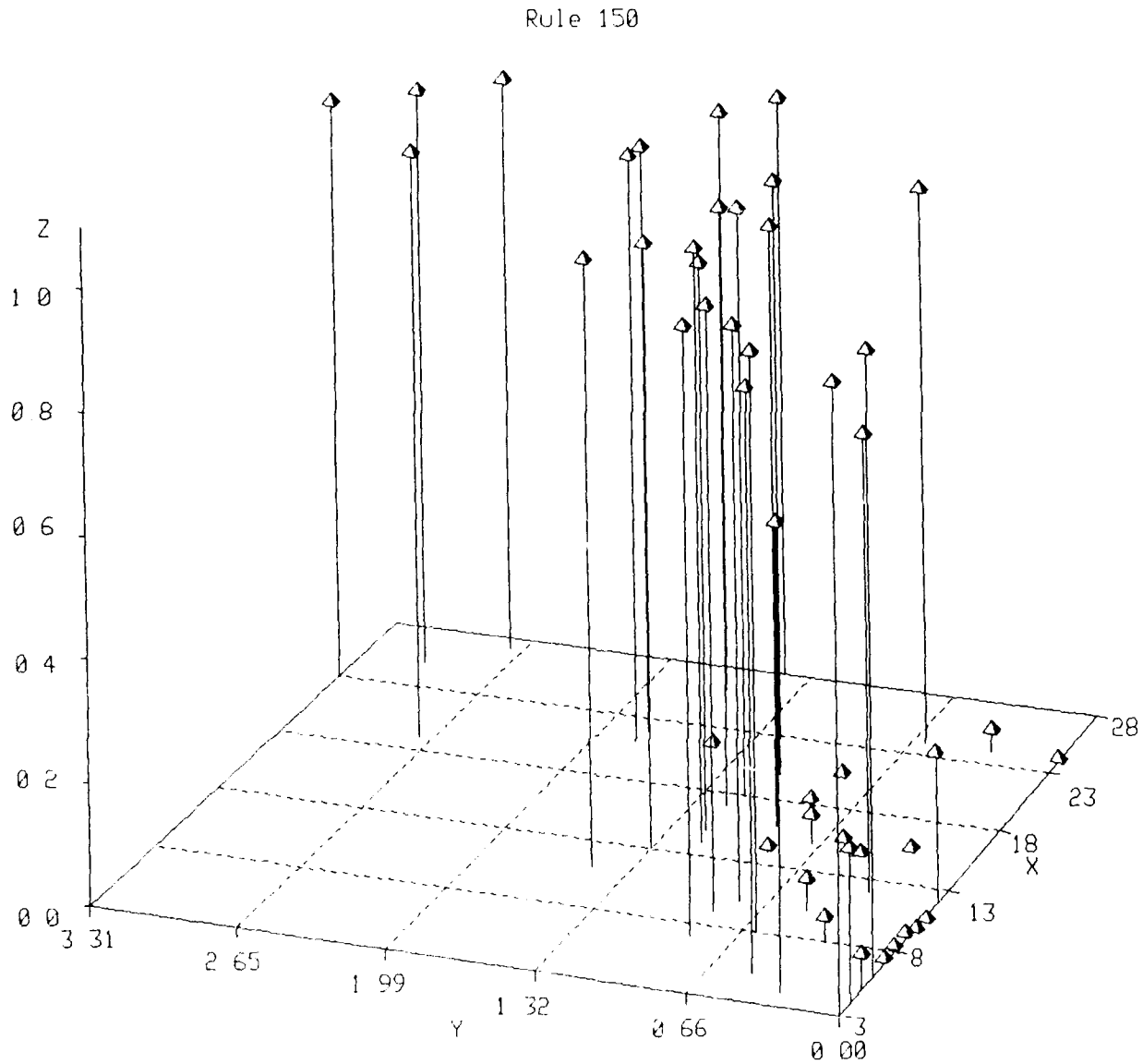
Rule 146



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D57. Attractor probabilities vs array size. Rule is defined in Appendix A.

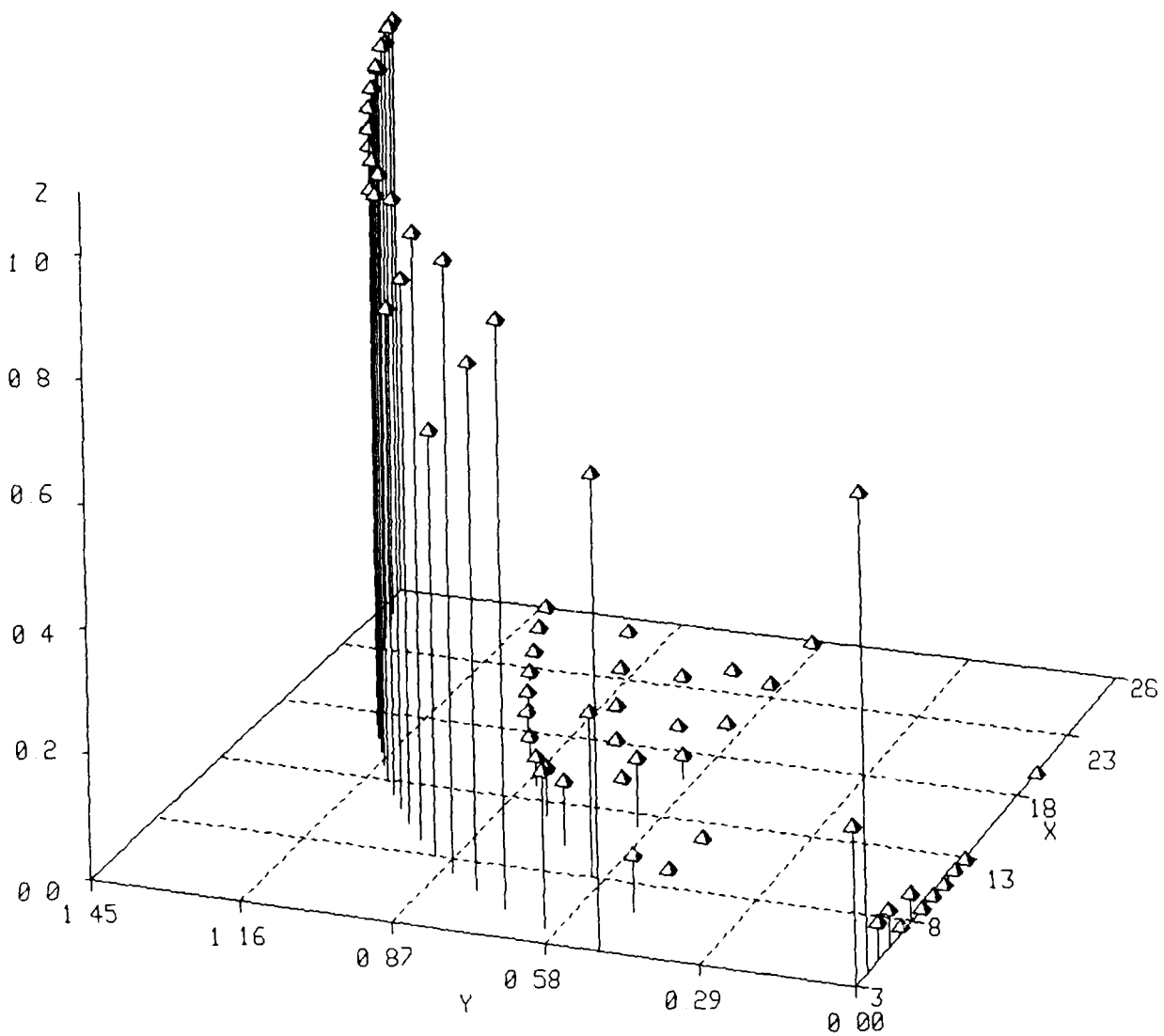


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D58. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 152

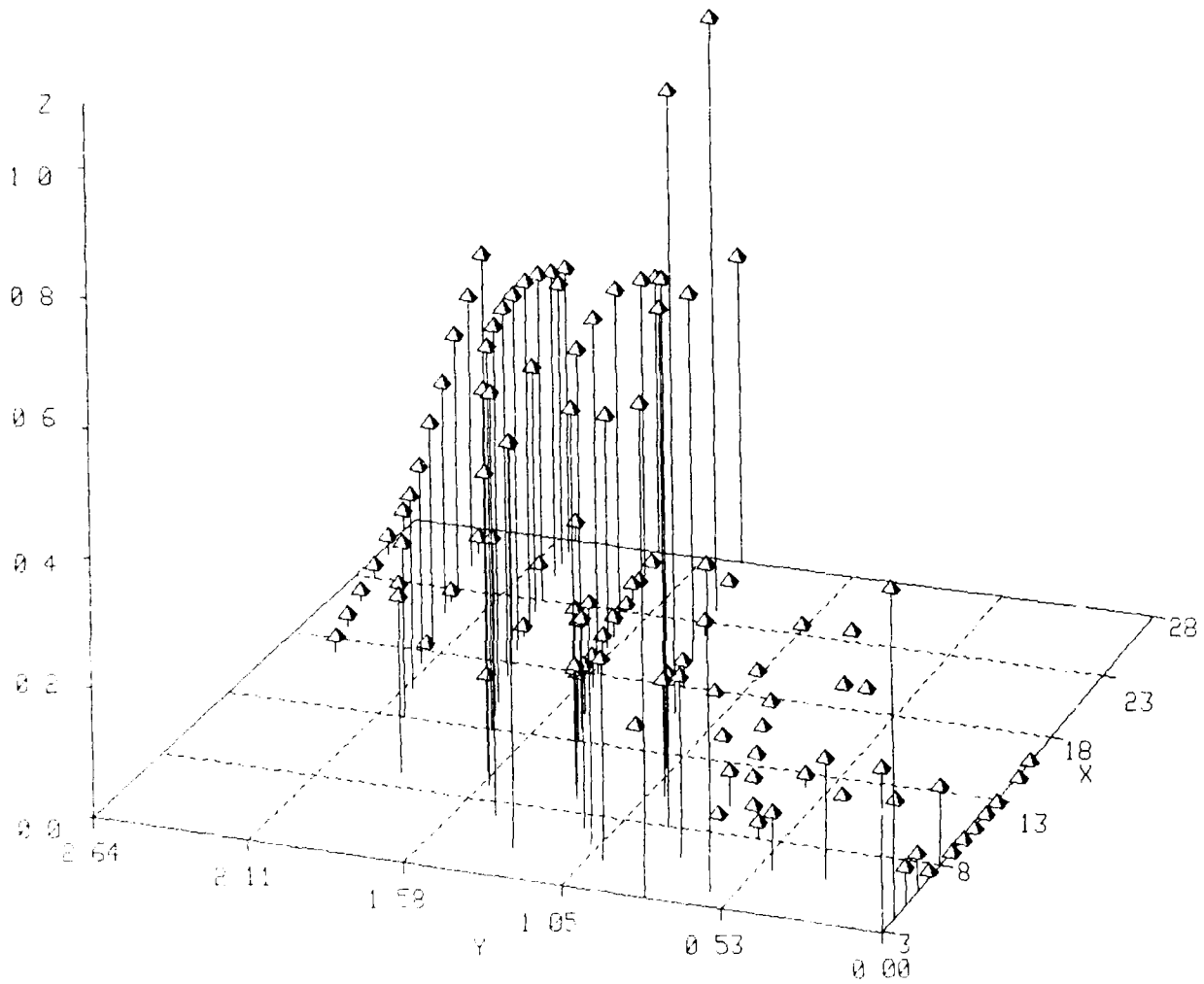


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D59. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 154

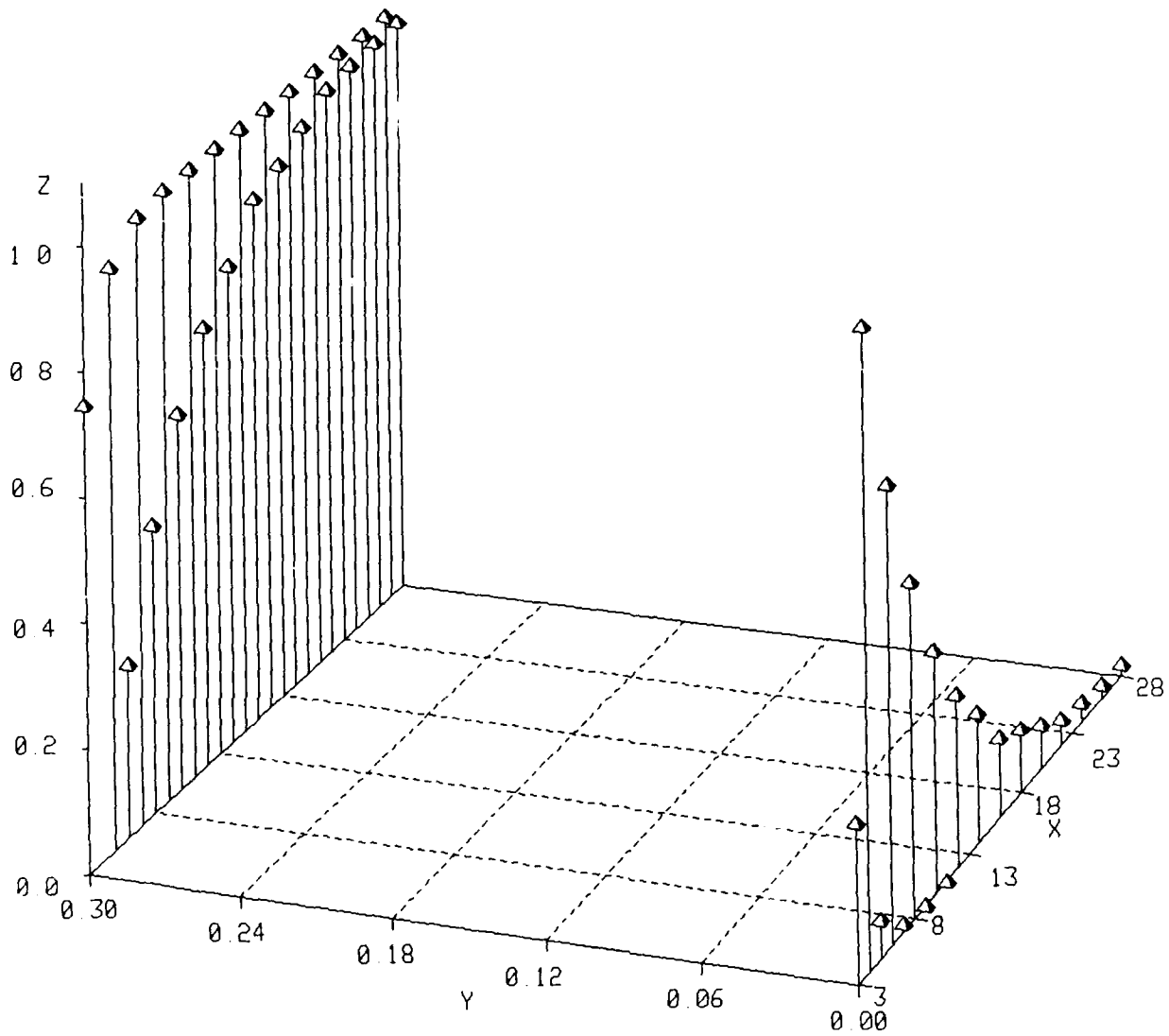


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D60. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 156

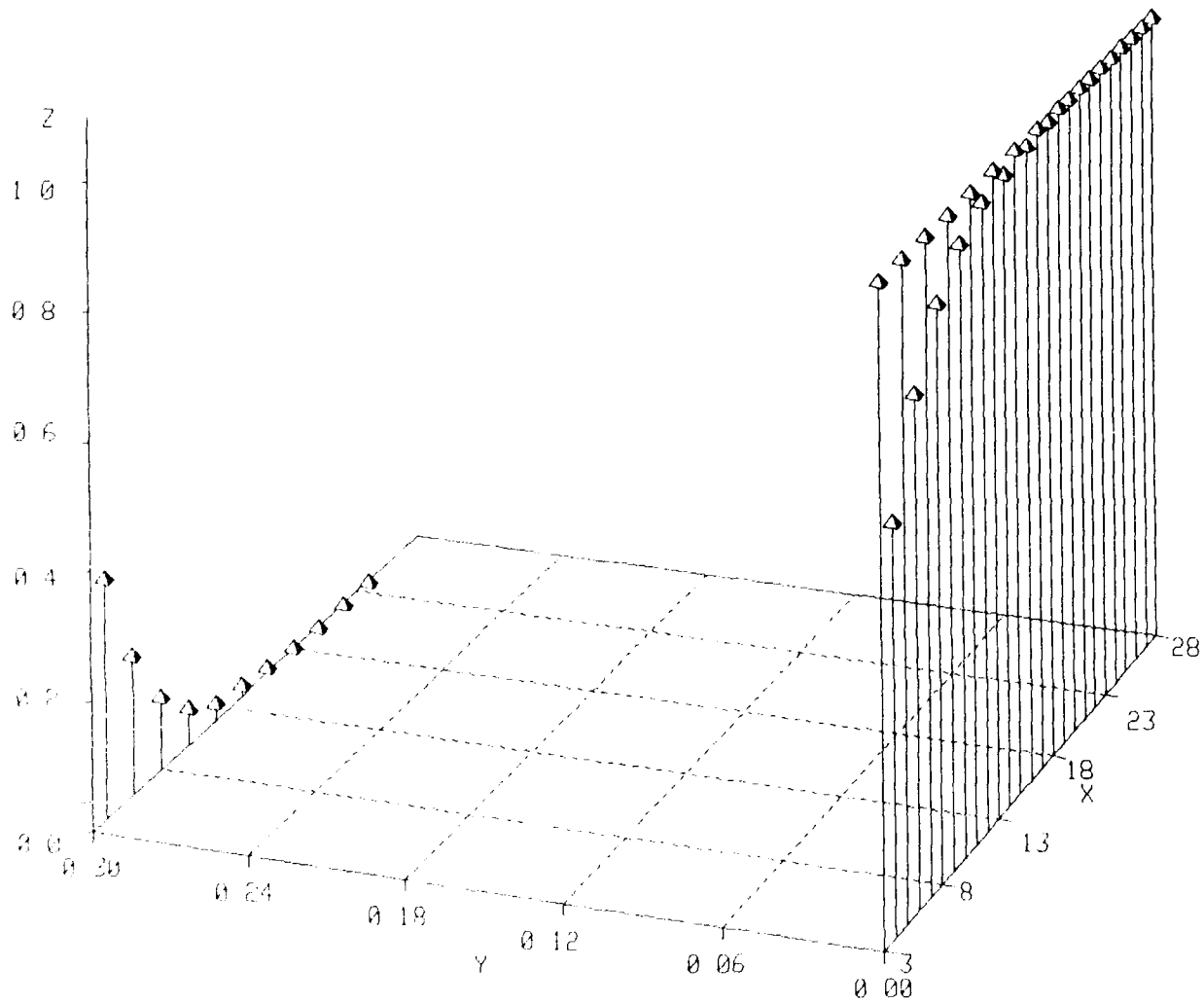


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D61. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 160

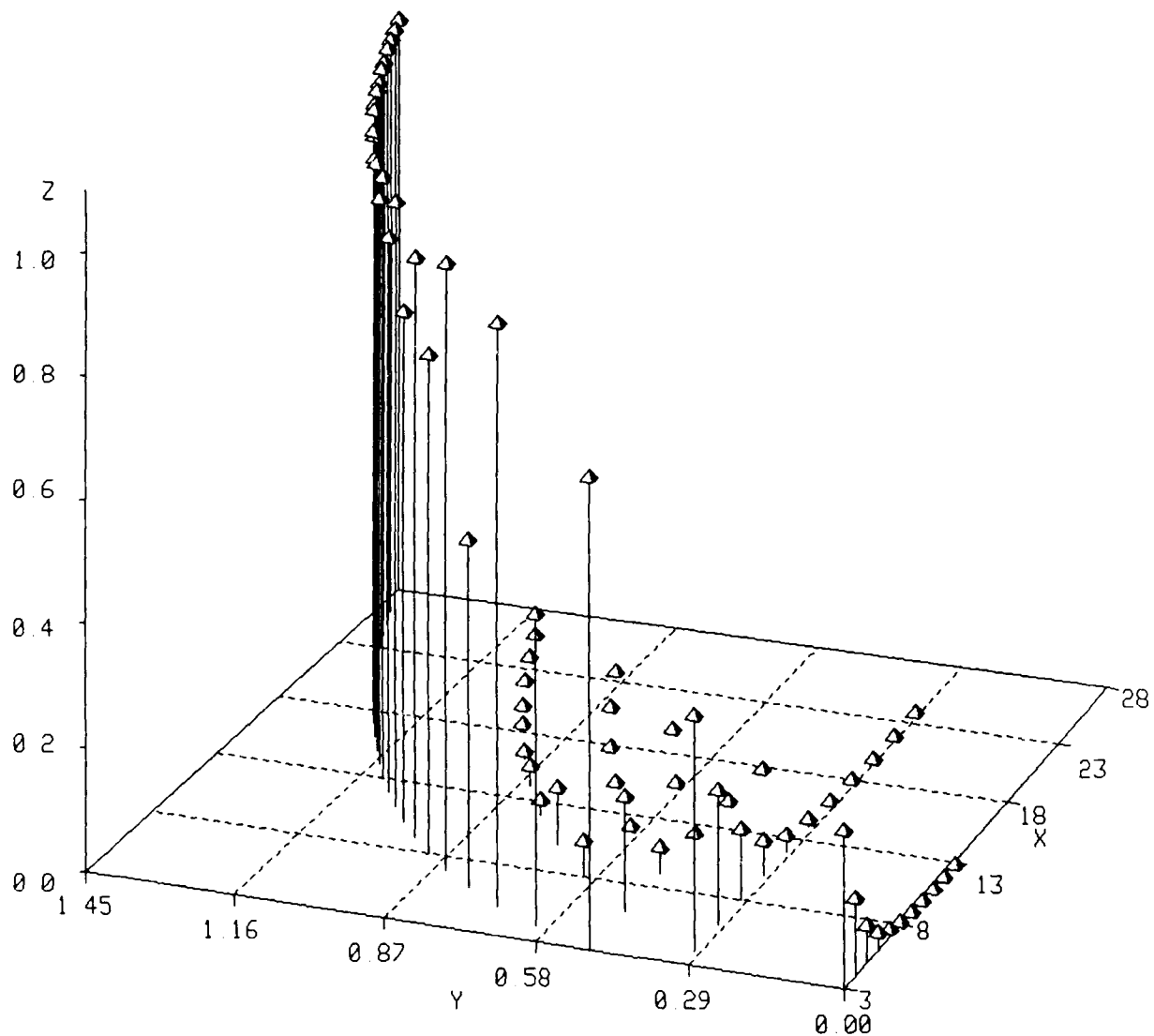


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D62. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 162

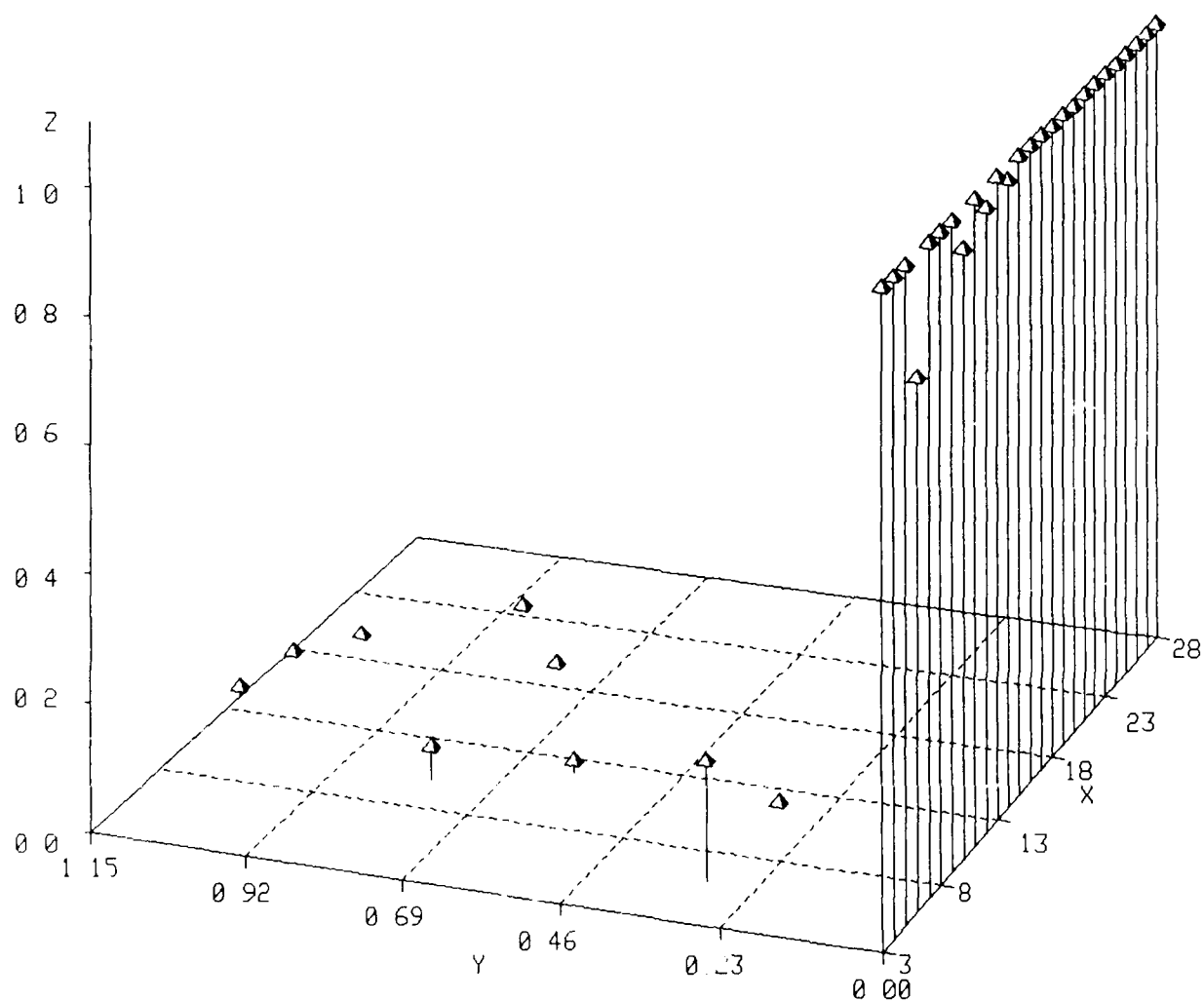


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D63. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 164

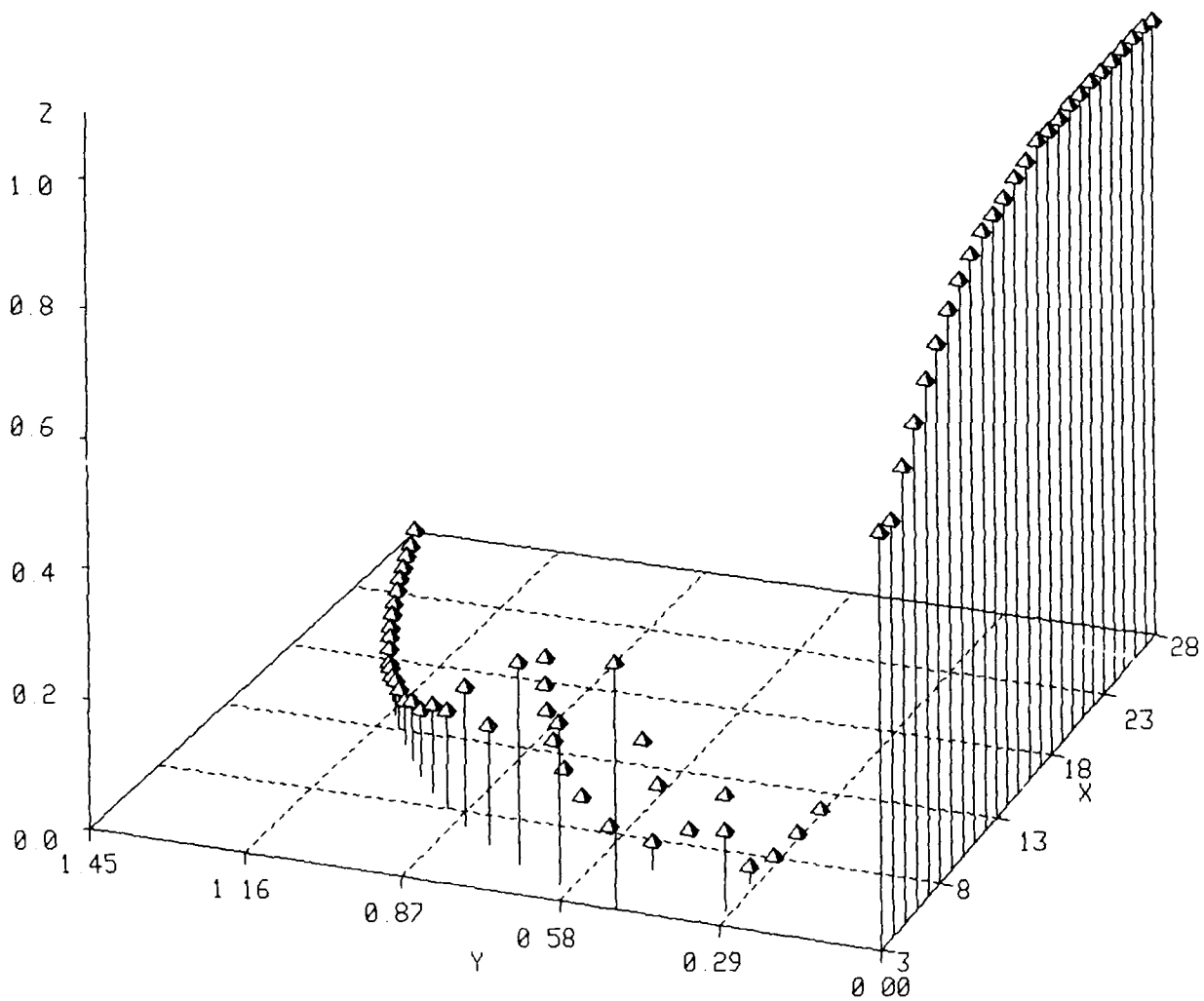


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D64. Attractor probabilities vs array size. Rule is defined in Appendix A.

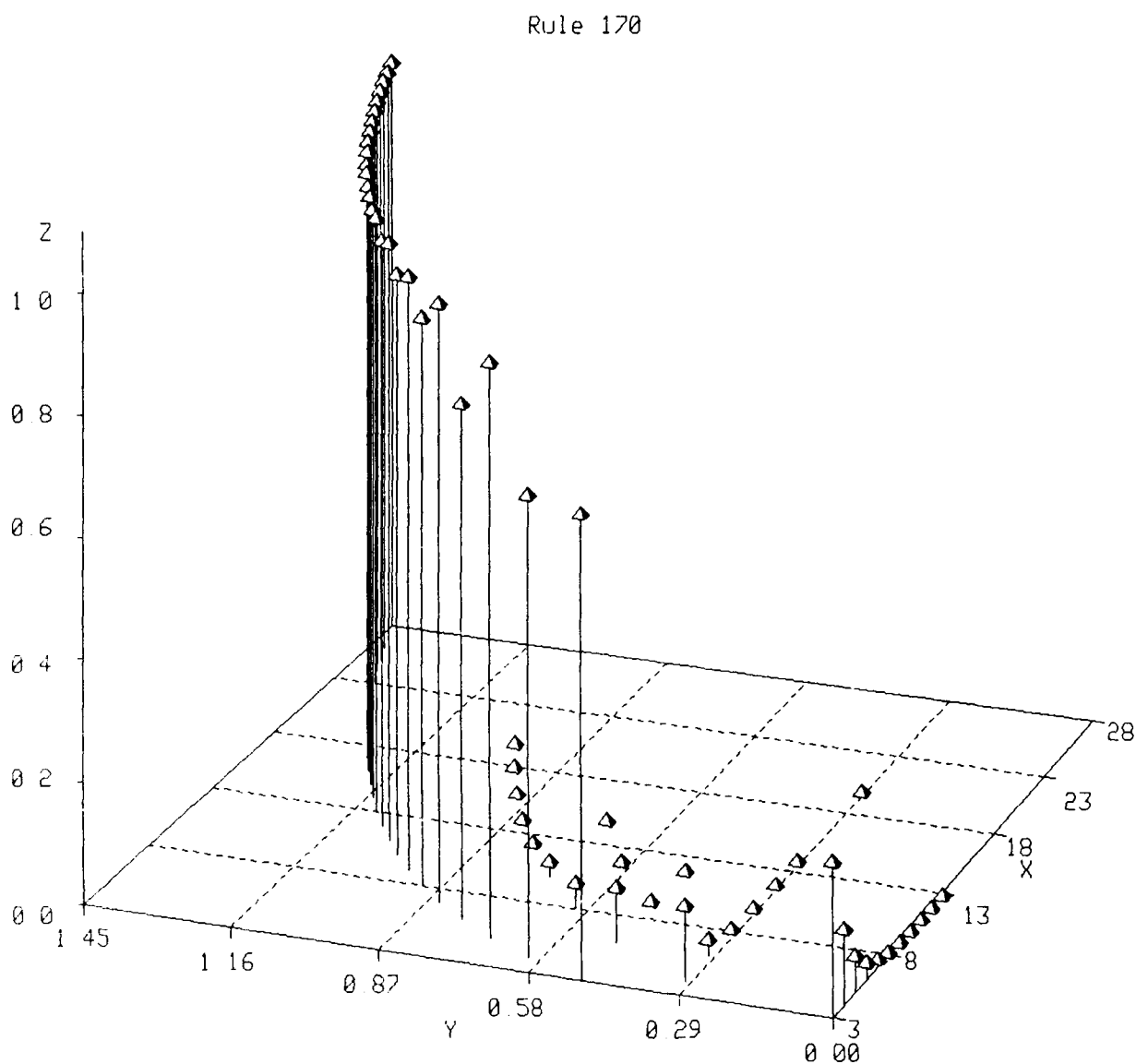
Rule 168



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D65. Attractor probabilities vs array size. Rule is defined in Appendix A.

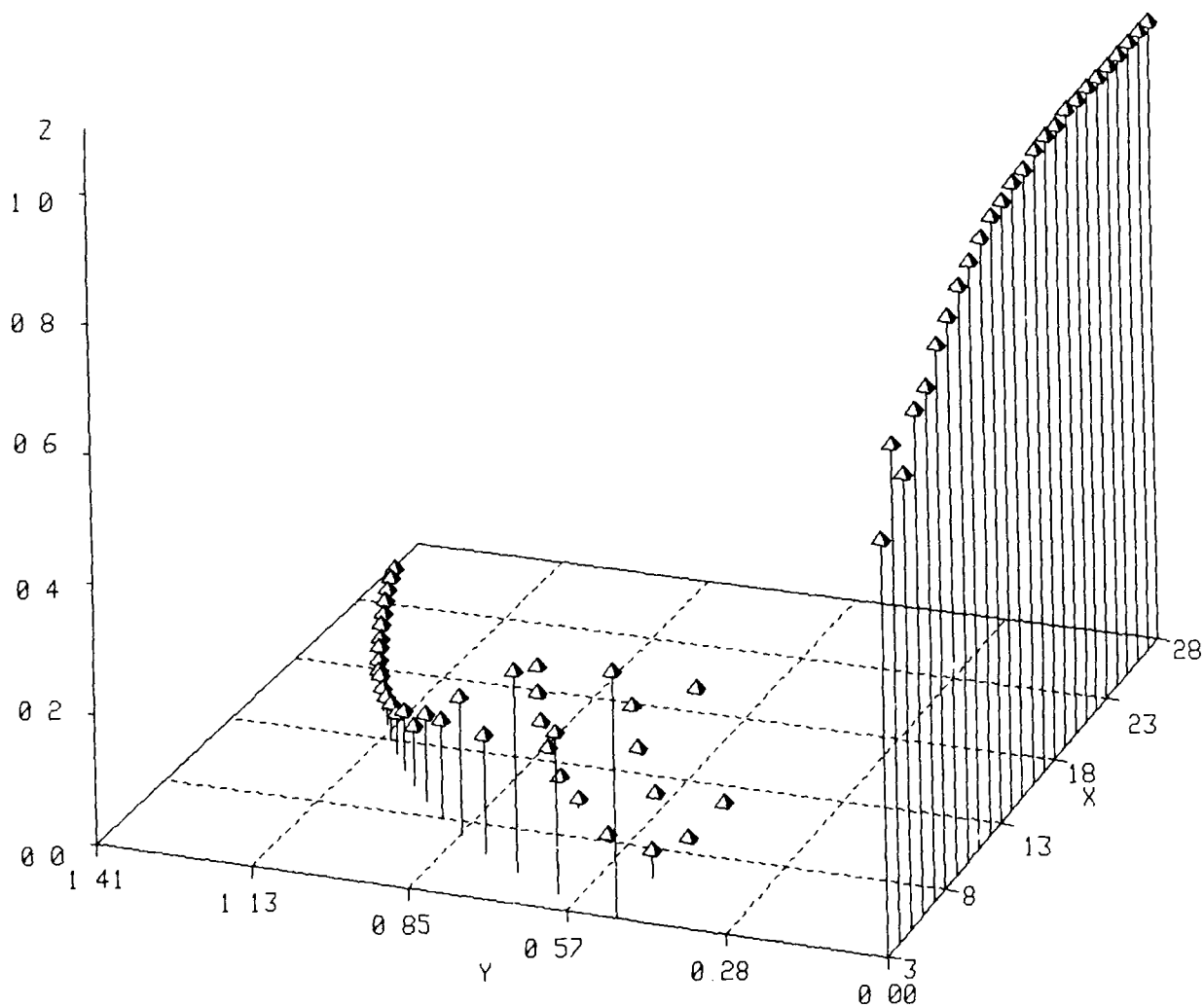


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D66. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 172

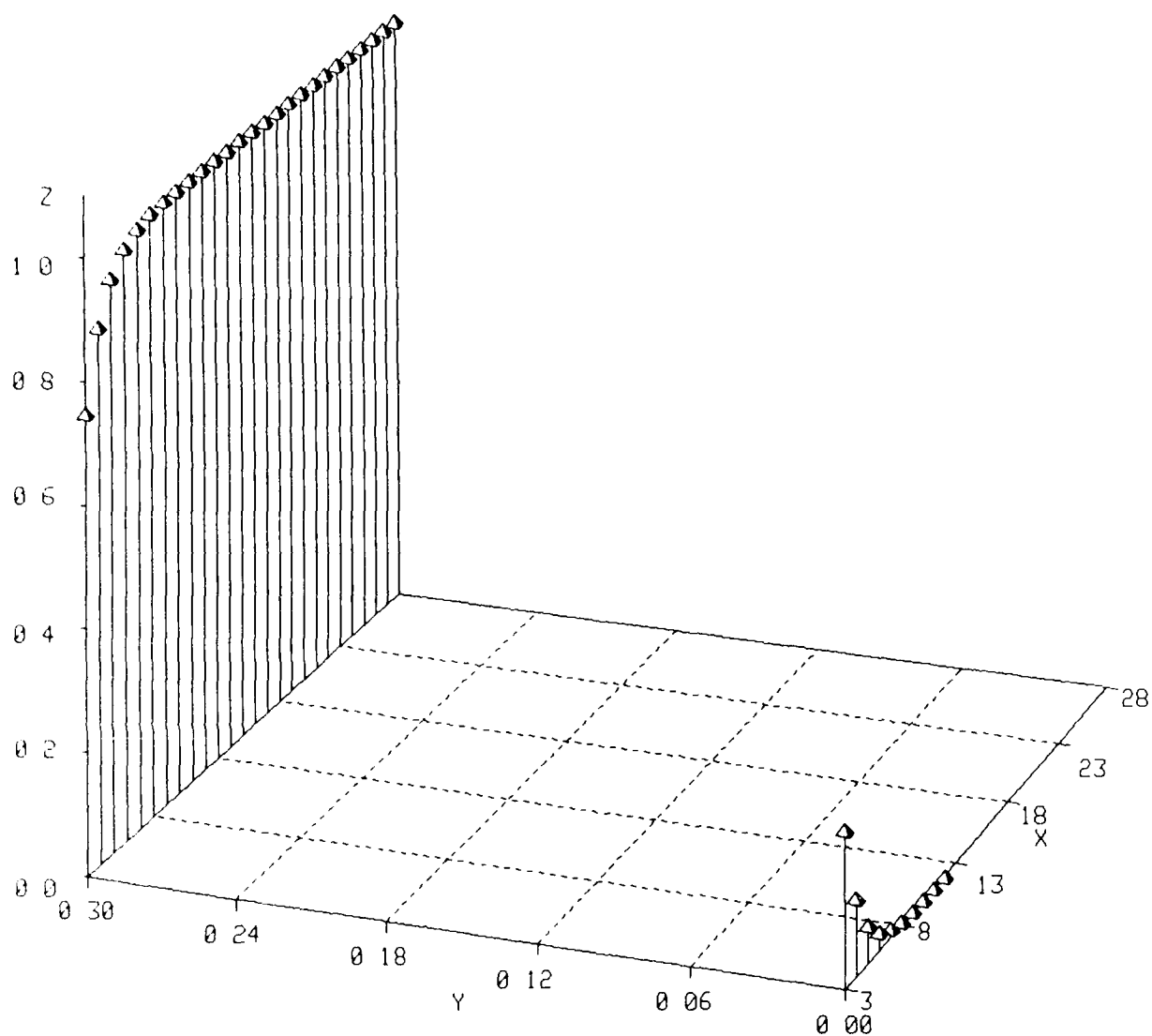


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D67. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 178

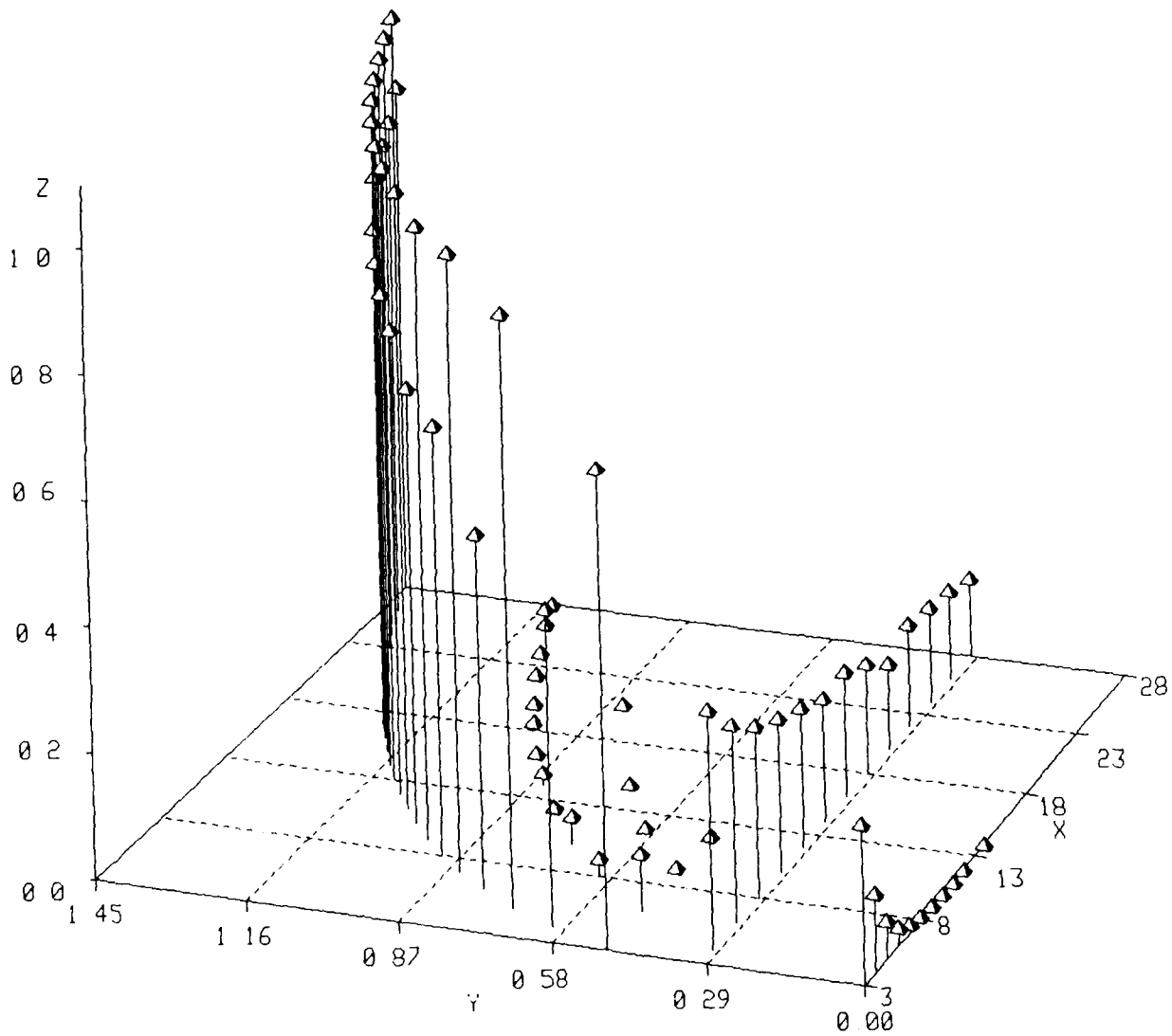


Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D68. Attractor probabilities vs array size. Rule is defined in Appendix A.

Rule 184



Legend

- X Number of Cells in Array.
- Y Log (Limit Cycle Length)
- Z Fraction of State Space Occupied by Attractor Basin.

Figure D69. Attractor probabilities vs array size. Rule is defined in Appendix A.

APPENDIX E

SENSITIVITY OF ONE-DIMENSIONAL CELLULAR AUTOMATA TO INITIAL CELL PATTERNS

APPENDIX E

SENSITIVITY OF ONE-DIMENSIONAL CELLULAR AUTOMATA TO INITIAL CELL PATTERNS

This section contains data on the amount of information in an initial cell pattern required to predict a particular attractor.

The strength of an attractor is measured by the volume of state space absorbed by the basin paths. It follows that a CA, characterized by large basins (few strong attractors), require relatively less information to be supplied to an initial cell pattern for the prediction of the final attractor observed. For example, a CA with a single attractor, such as rule 255, requires no preselection of cell state patterns to predict with certainty the future dynamics of the automaton. However, rules including Nos 18 , 22 and 45 are characterized by numerous attractors of various basin sizes. In these latter examples, predicting a particular limit cycle requires careful specification of the starting cell configuration.

The normalized measure of the input sensitivity of the rule, can be expressed as

$$I_{SR} = \frac{\sum_{i=1}^n V_i \ln V_i}{\ln(n)}$$

where V_i is the probability of observing attractor i from a random starting cell pattern, and n is the number of attractors observed for the given rule and array size.

In simulating the model dynamics, the ends of the linear array of cells were coupled together so that the cells formed a closed loop. That is, the edge cells are actually nearest neighbors in the simulations. Of the 256 possible nearest-neighbor cell rules, only 88 rule operations are independent. Since for example, rules 14, 84, 143, and 213 lead to identical attractor spectra, only Rule 14 is included in the plots. In addition, several independent rules lead to constant sensitivity factors.

We did not include any rule which was characterized by a single attractor. The value of ISR is zero for these rules. Rule 204 is an example of an independent rule which merely replicates the starting cell pattern over time. The input sensitivity of such rules is trivial to predict. Refer to the Appendices A and B for a complete list of independent rules and their attractor sets.

The rules were classified according to whether the average input sensitivity was a decreasing, increasing, or oscillatory function of array size.

The input sensitivities fall into three simple classes of behavior, distinguishable by the various attractor types observed.

Type I1 Sensitivity : ISR decreases with array size on average.

Type I2 Sensitivity : ISR increases with array size on average.

Type I3 sensitivity : No average correlation of ISR with array size.
Sensitivity oscillates with array size.

The following assignments were made from an inspection of the sensitivity data.

Type I1 Rules : 2, 3, 5, 6, 7, 10, 13, 15, 23, 24, 27, 28, 29, 31, 34, 35,
37, 38, 40, 42, 46, 50, 56, 60, 150, 152, 156, 160, 162,
164, 170, 172, 178

Type I2 Rules : 25, 45, 94, 108

Type I3 Rules : 9, 11, 14, 18, 22, 26, 30, 41, 43, 54, 57, 58, 62, 73, 154,
146, 106, 110, 122, 184

The input sensitivity of Type I1 rules numbered 13, 50, 77, and 178 were found to decrease monotonically in sensitivity for all array sizes studied. these rules are characterized by only two attractors of length one

(1) and two (2) states. For small array sizes, the two attractors are equally probable. As a result, ISR for small array sizes is nearly unity. As the array size is increased, one attractor increases its domination of state space until only one attractor is observed.

Rules 25, 45, 94, and 108 show a slow increase in their ISR parameters with array size. Rule 25 is composed primarily of attractor lengths which are integral multiple of array size. As array size increases, more evenly divisible cycle lengths are permitted. This leads to an overall increase in the information required to predict a given attractor. Rule 108 displays the inverse behavior of the monotonic rules discussed above. For this rule, the two possible attractors tend to share equally the exponentially increasing volume of state space as array size grows. Finally, Rule 45 is the most complex rule observed within the class. This rule is characterized by a set of uncorrelated attractor sizes, some of which are nearly fully ergodic. It is not possible to explain simply the behavior of this rule in general terms.

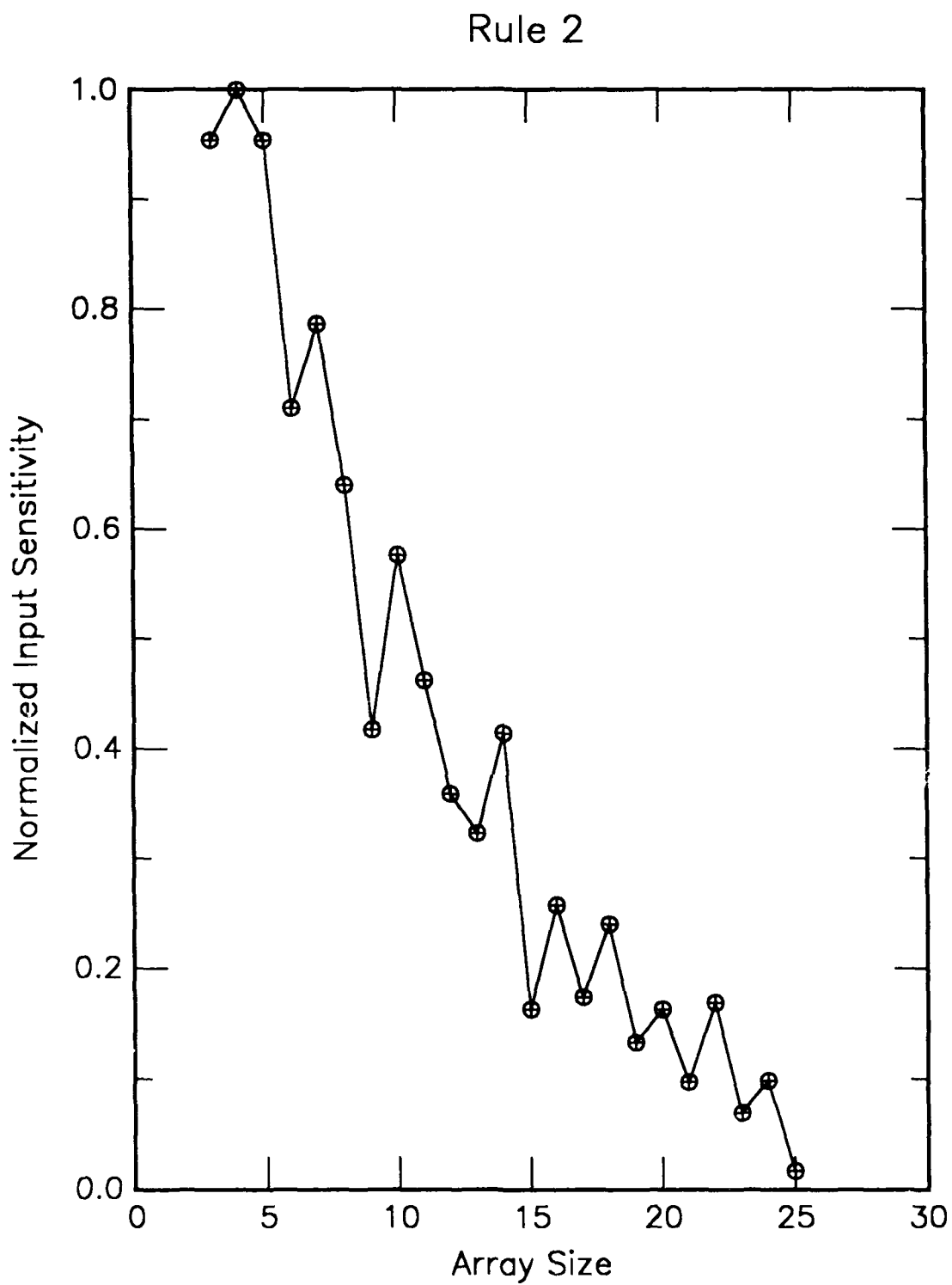


Figure E1. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

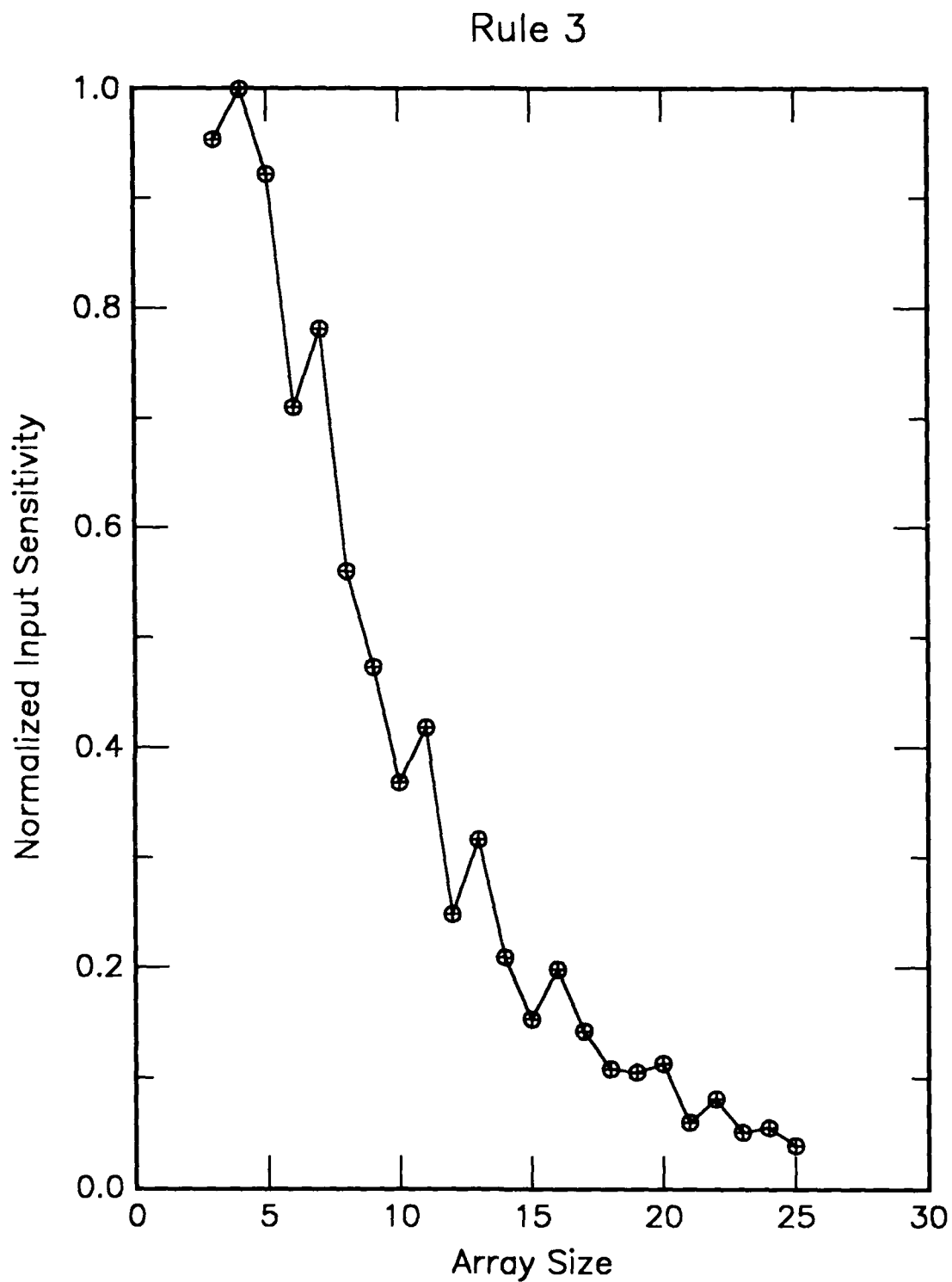


Figure E2. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 5

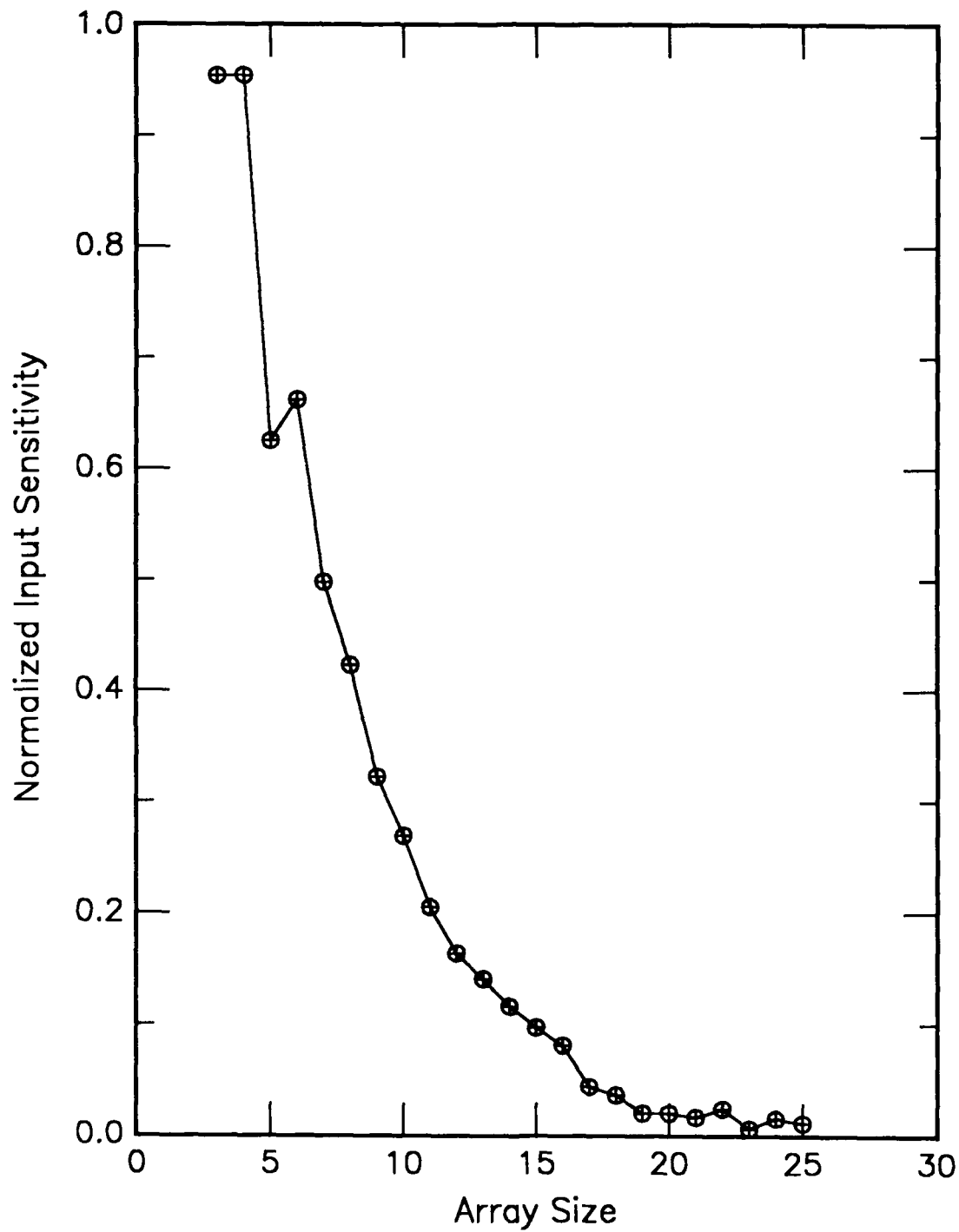


Figure E3. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 6

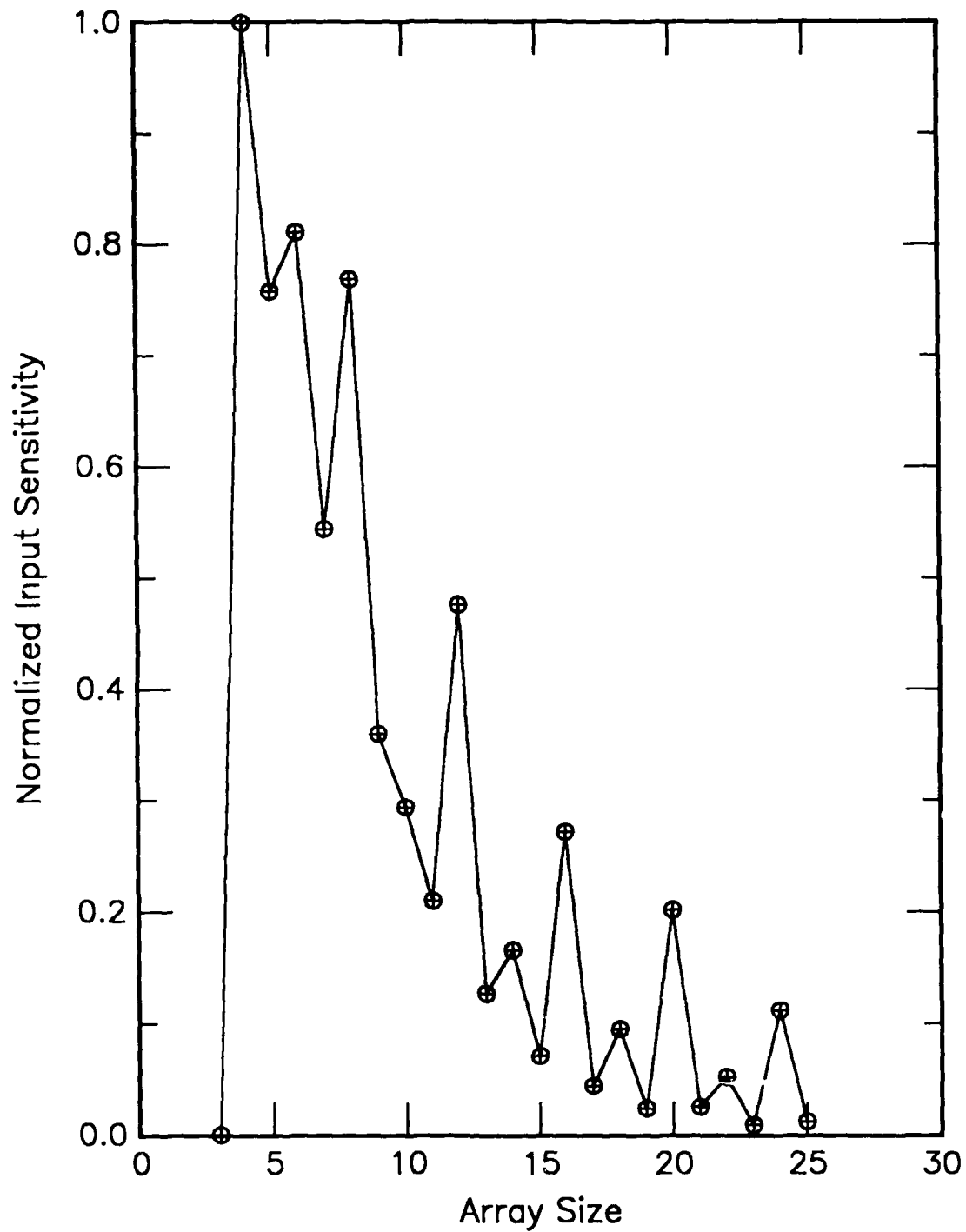


Figure E4. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 7

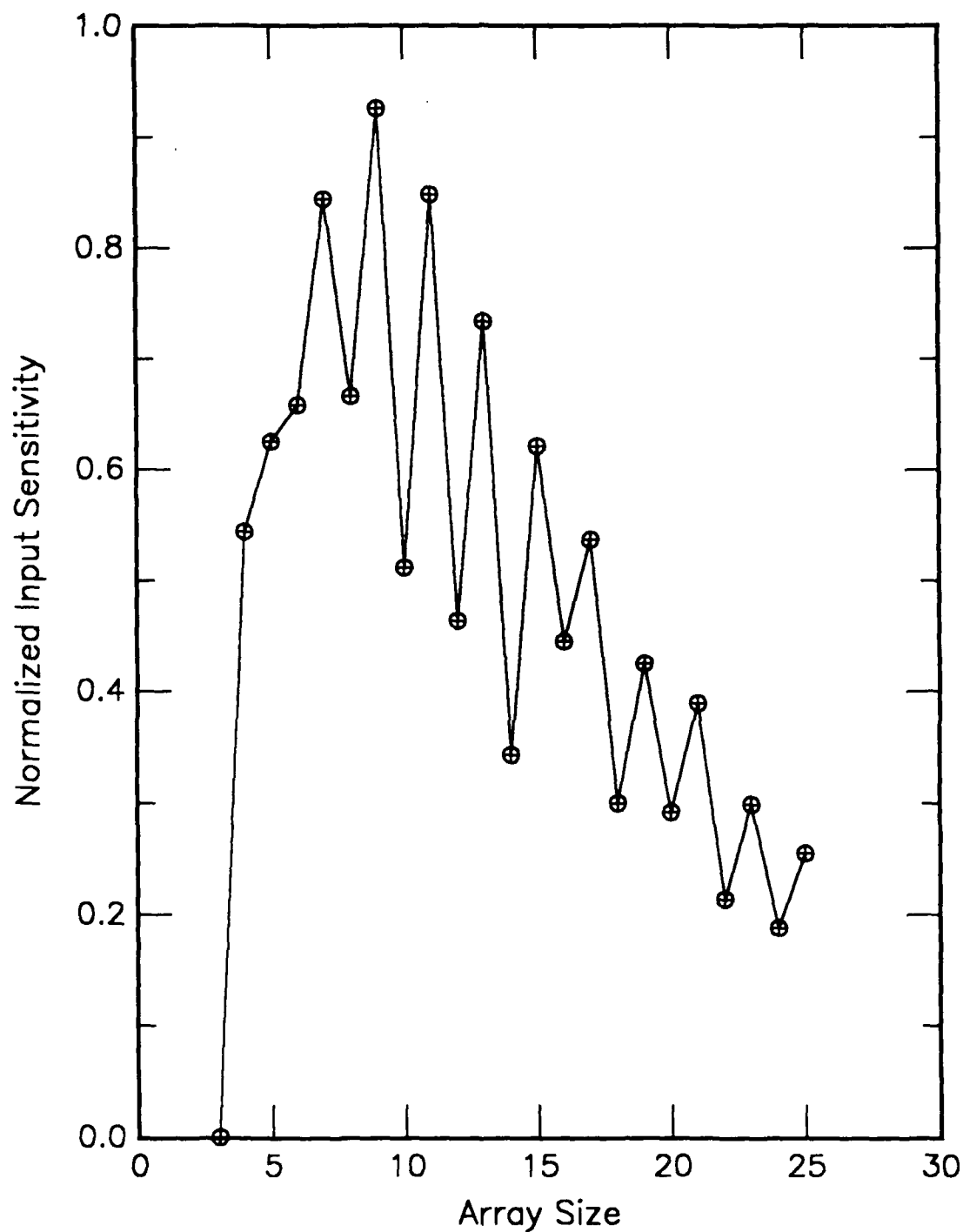


Figure E5. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 9

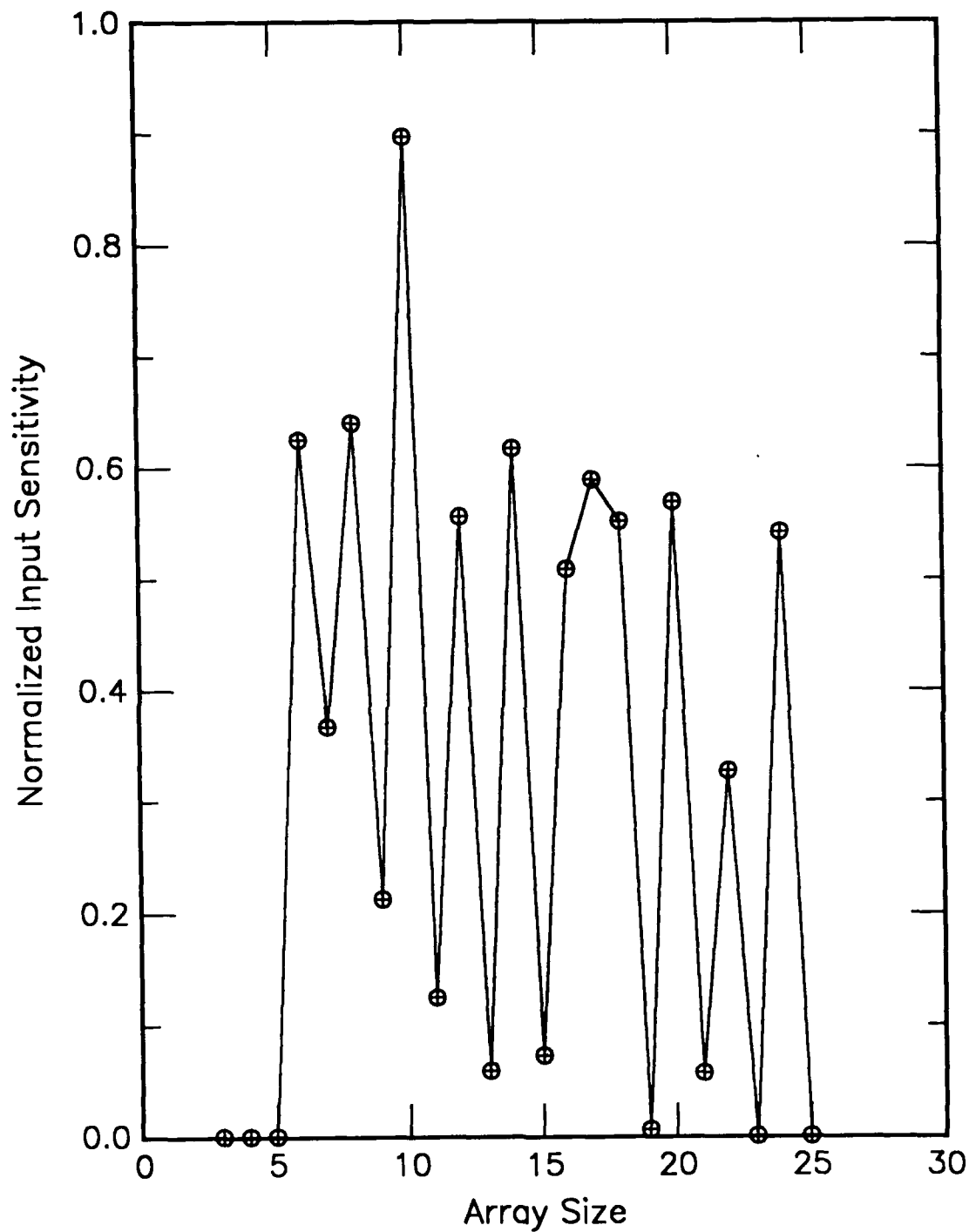


Figure E6. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 10

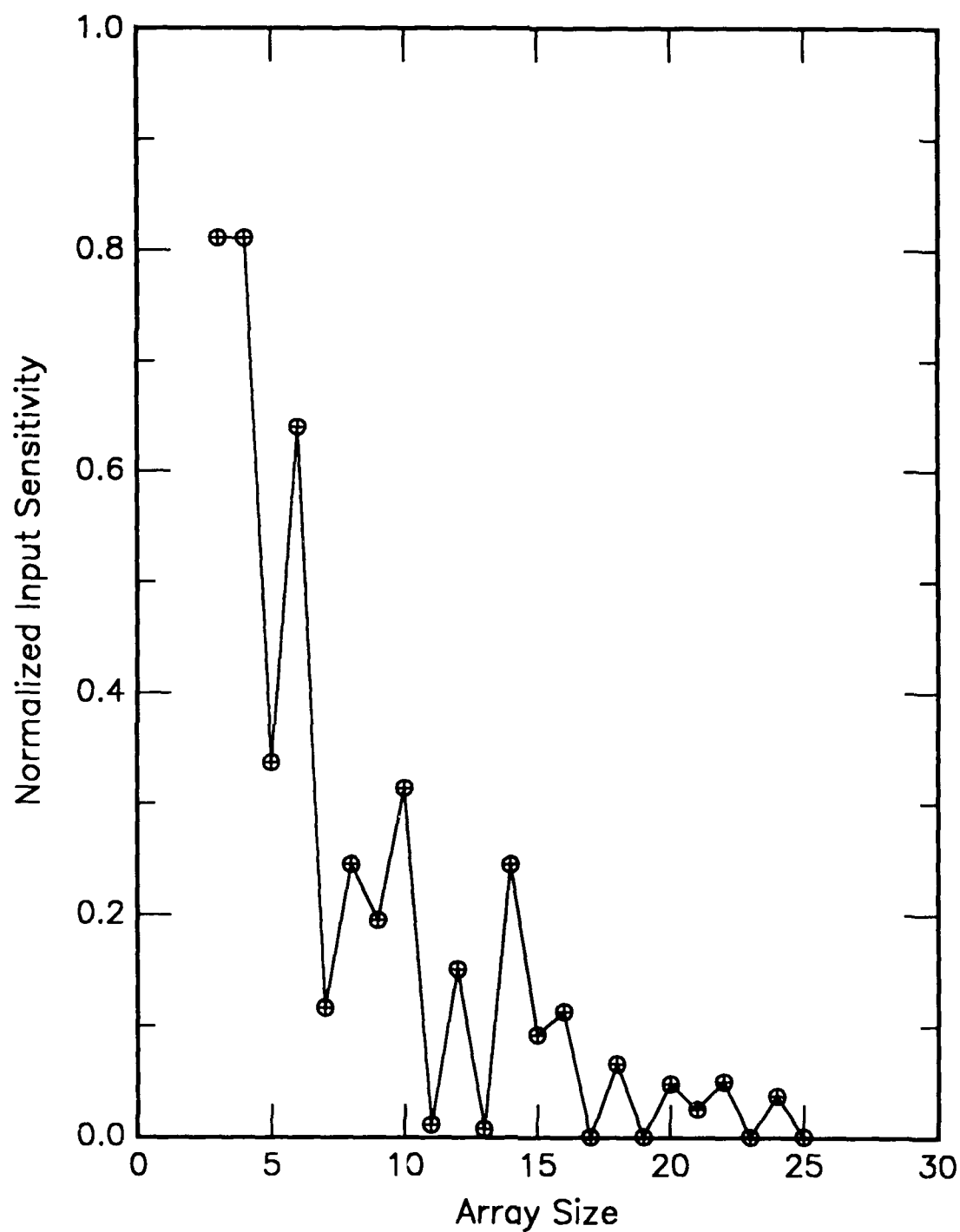


Figure E7. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

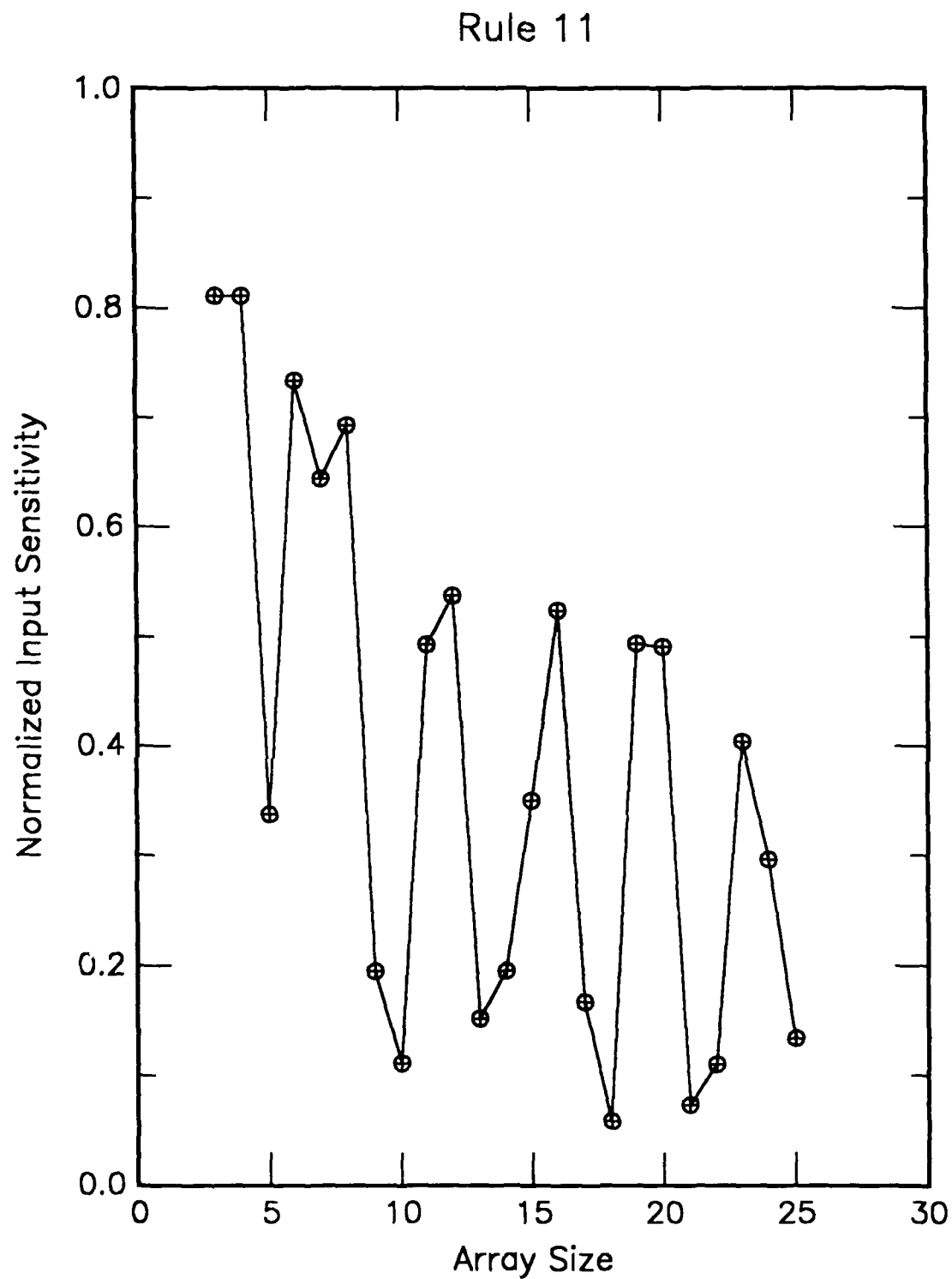


Figure E8. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 13

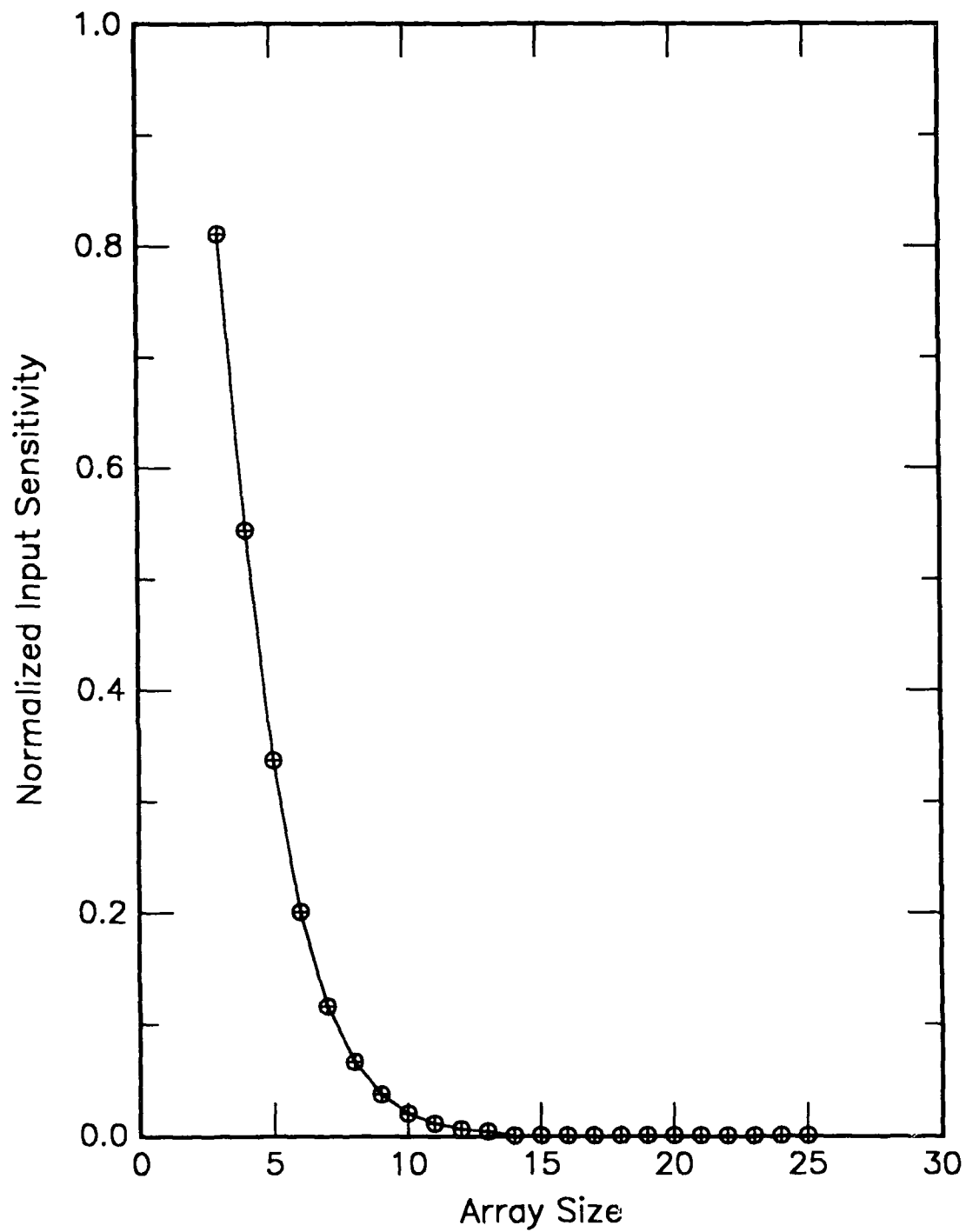


Figure E9. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

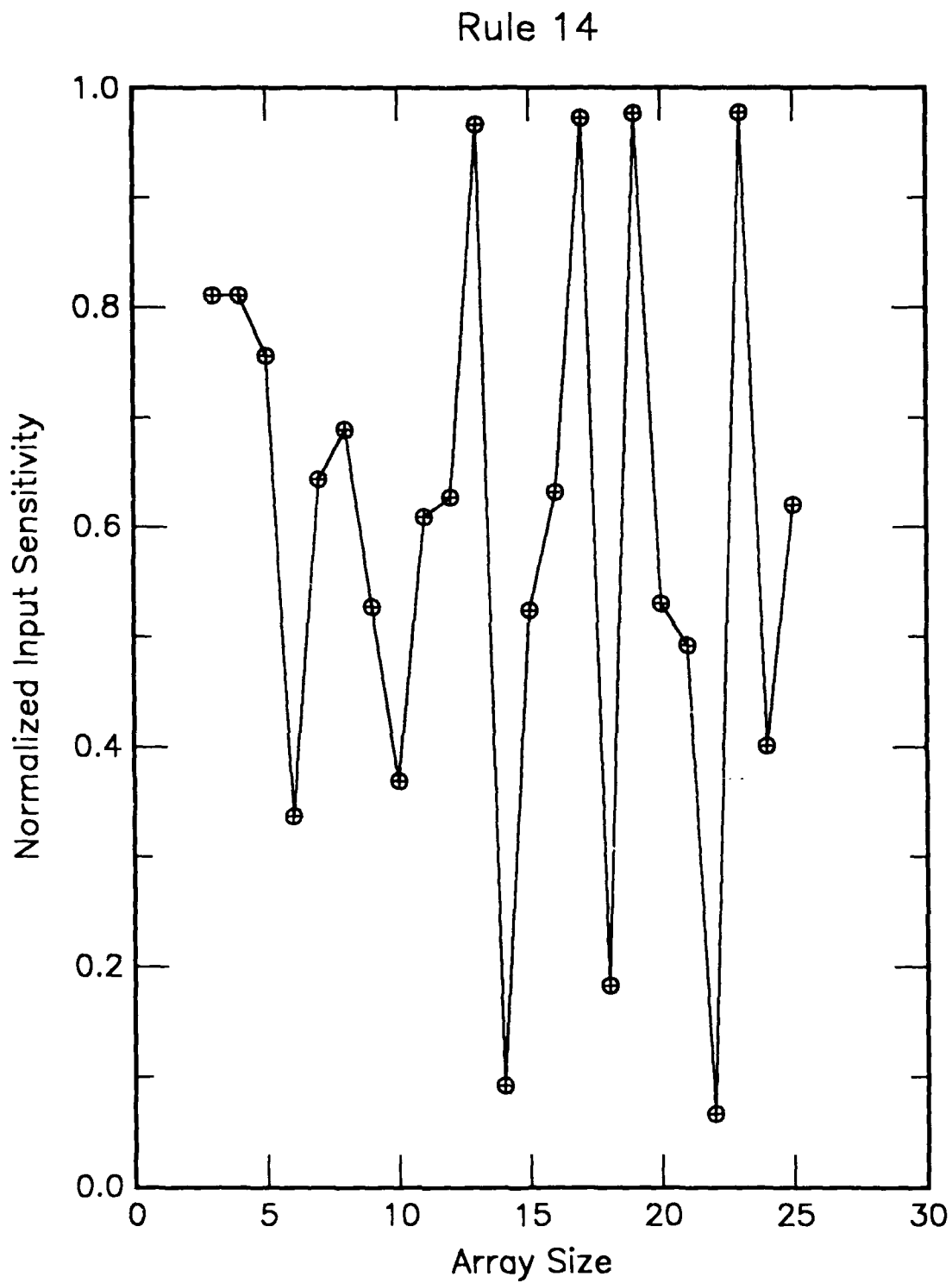


Figure E10. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 15

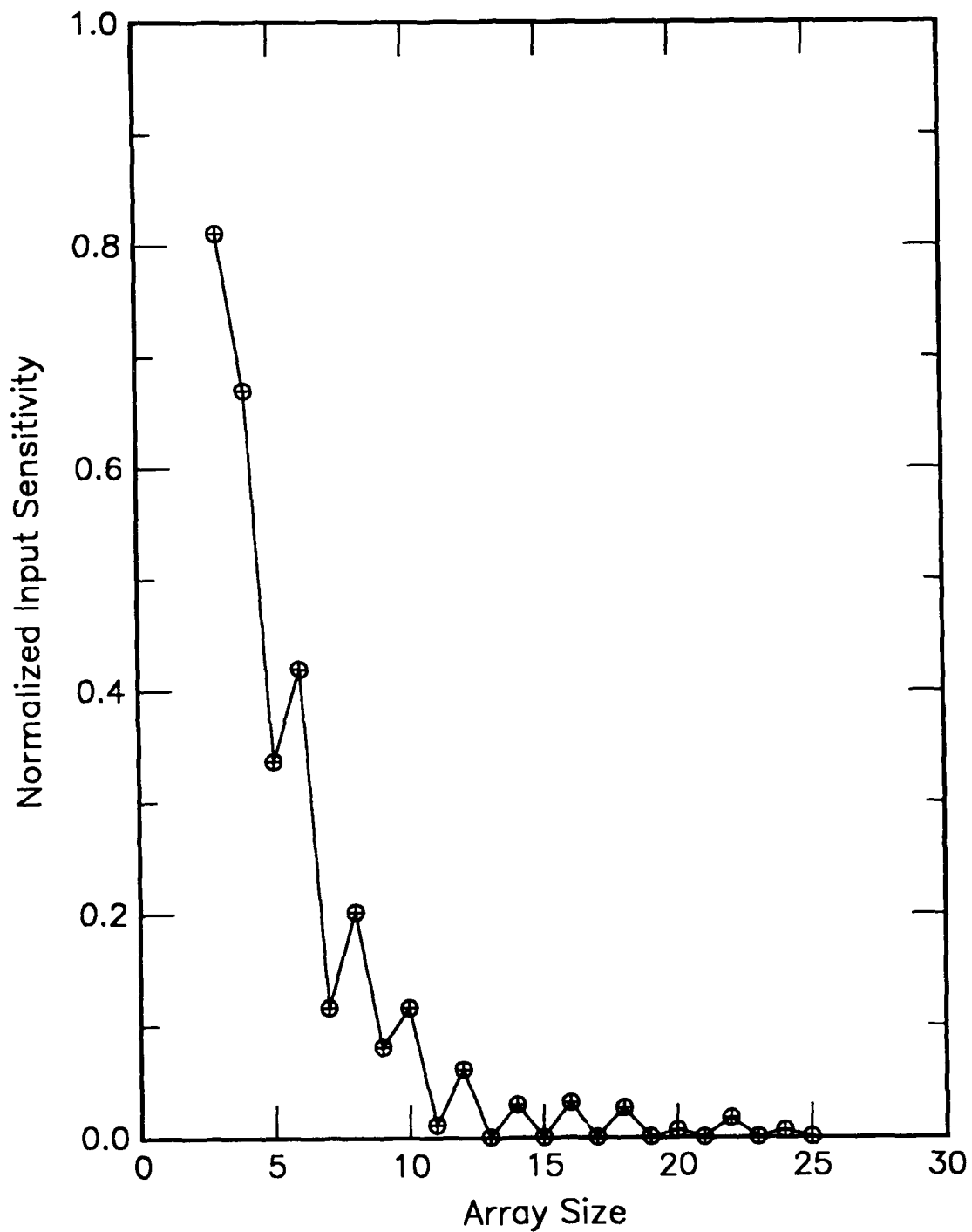


Figure E11. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 18

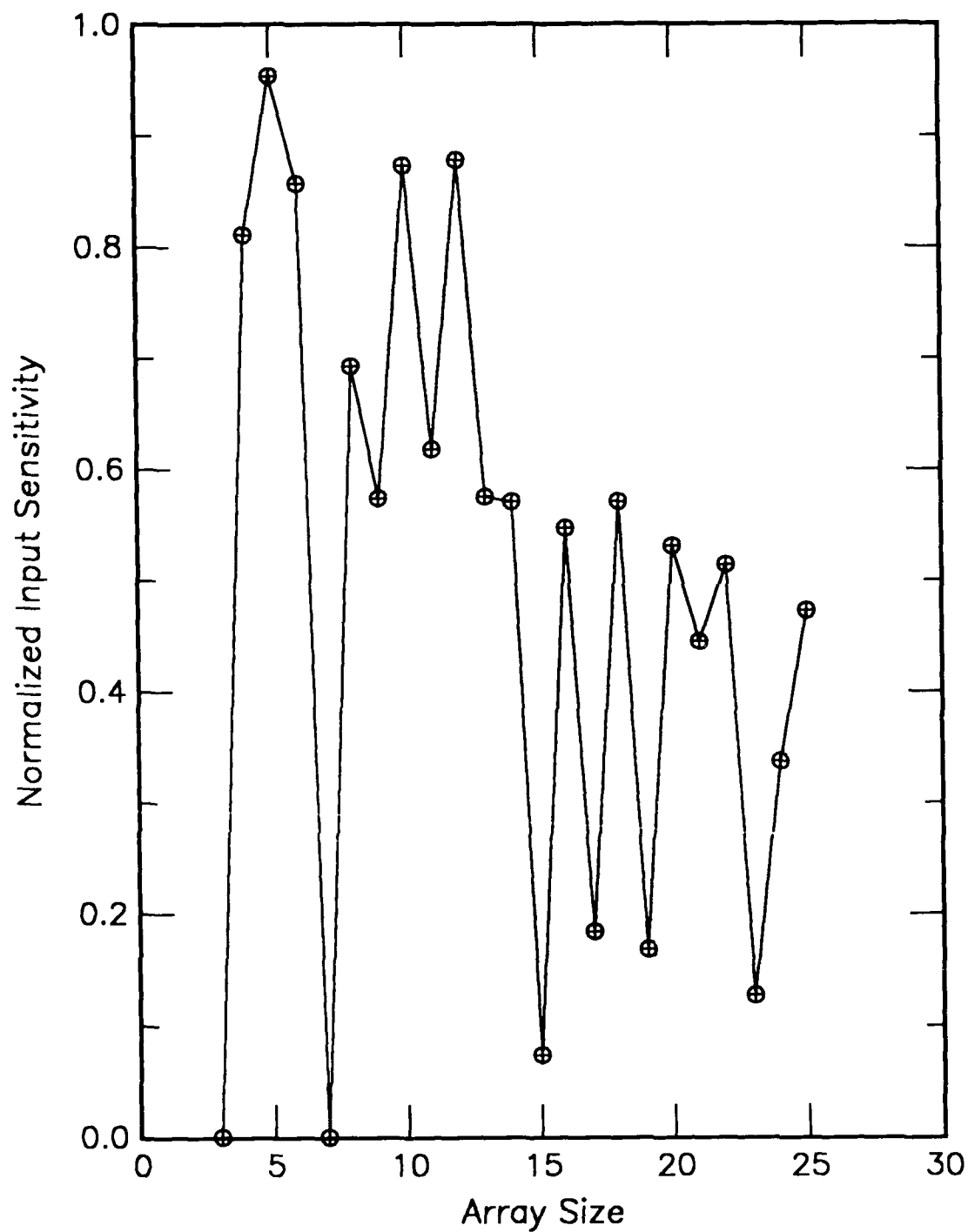


Figure E12. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

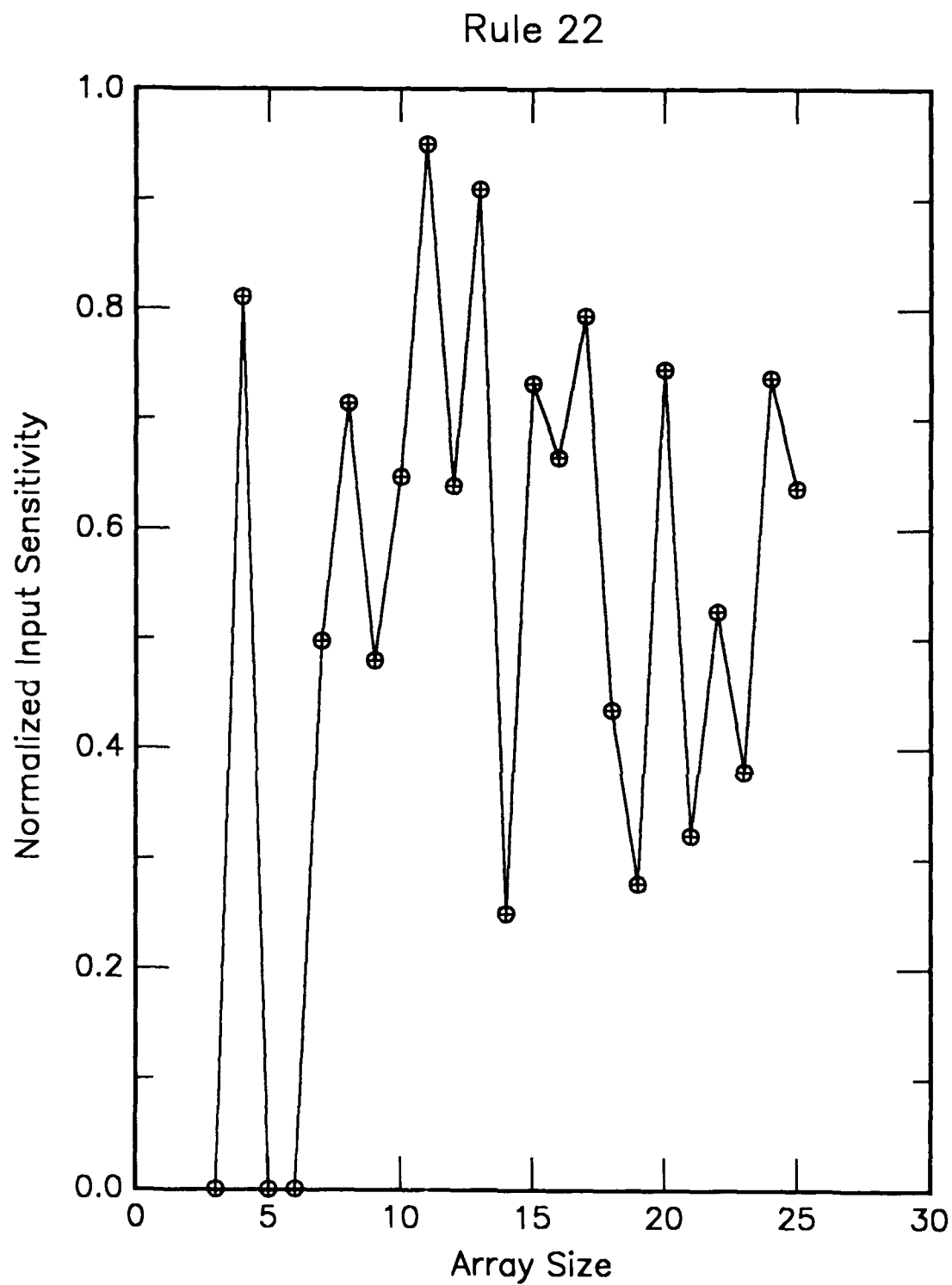


Figure E13. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 23

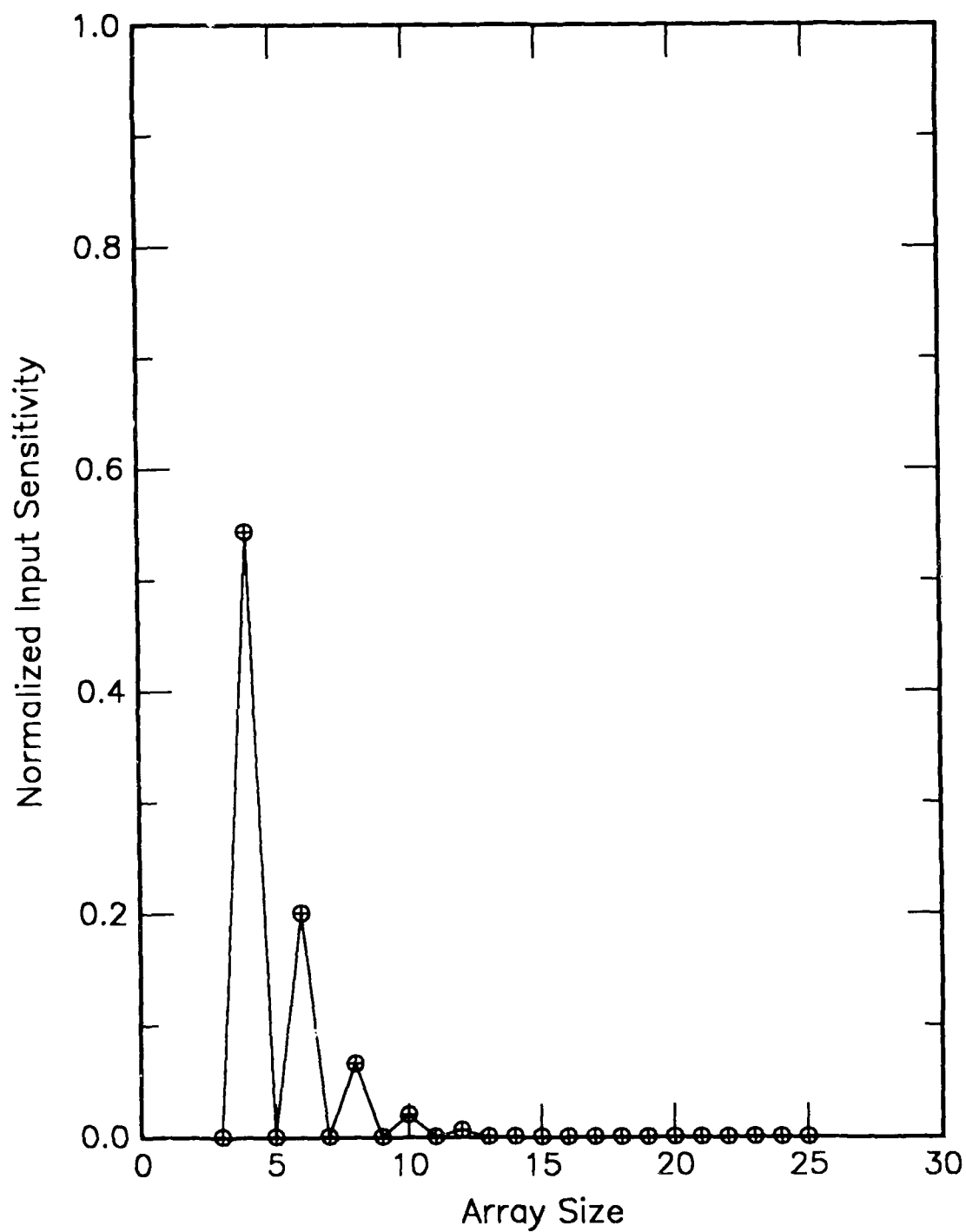


Figure E14. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 24

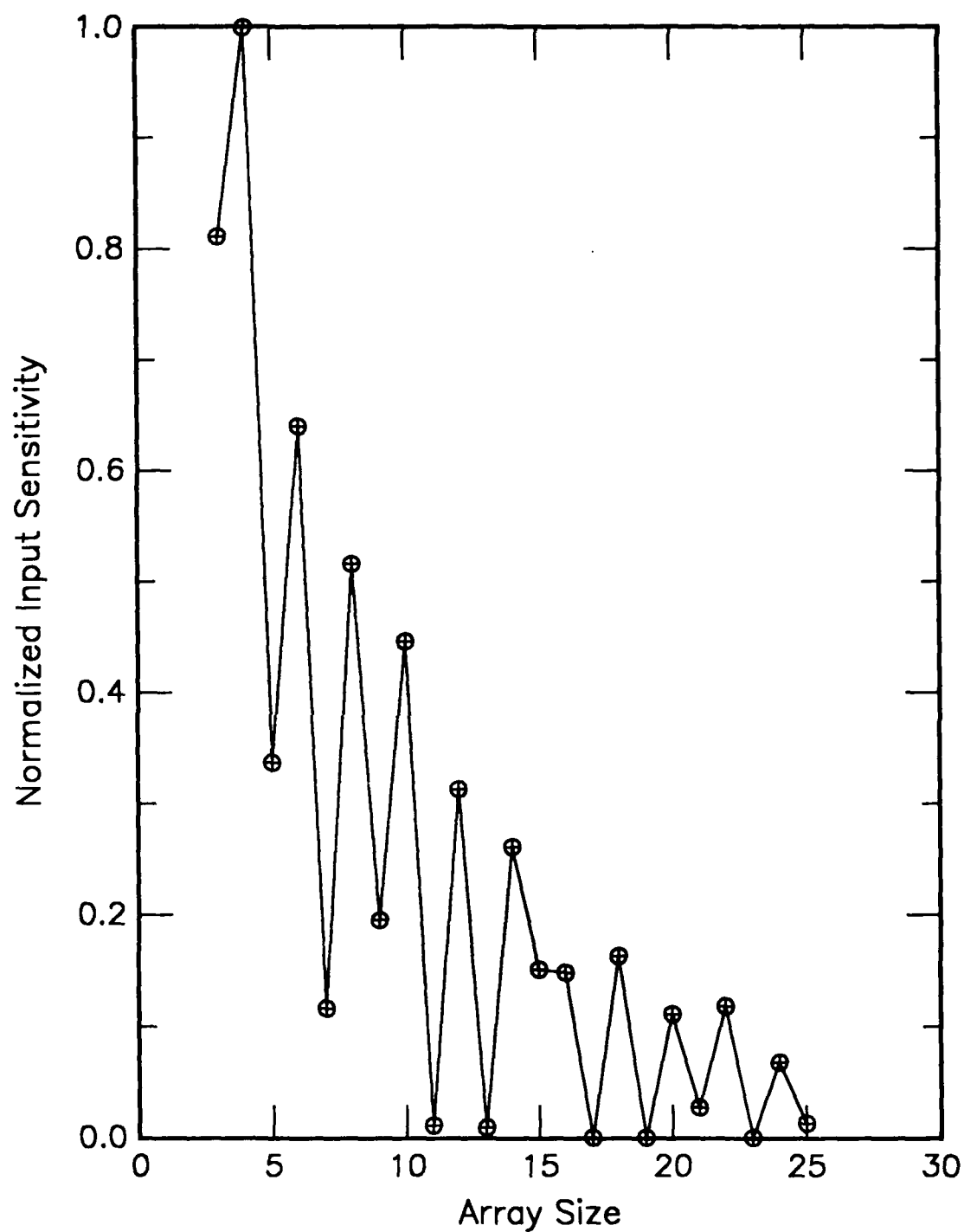


Figure E15. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 25

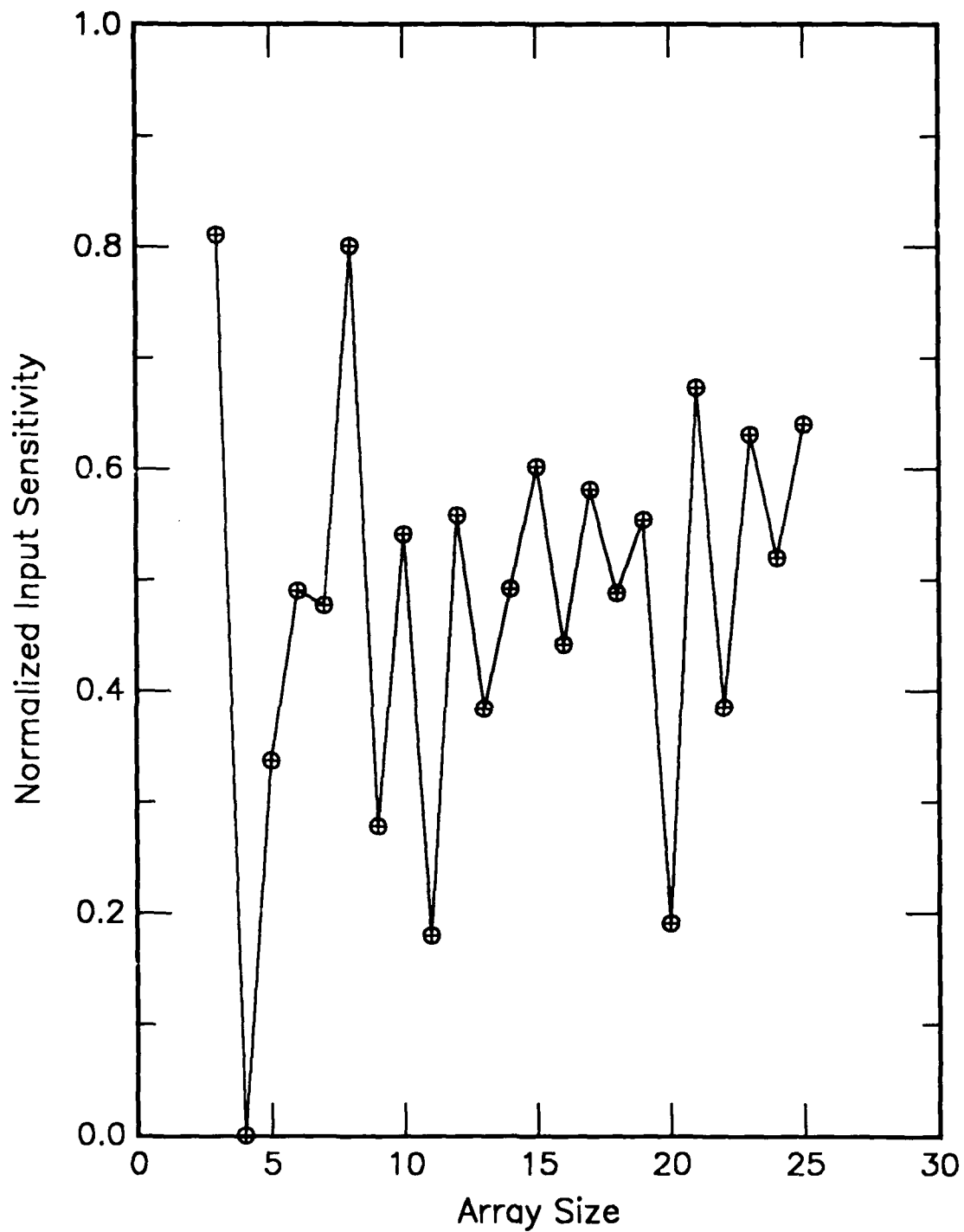


Figure E16. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 26

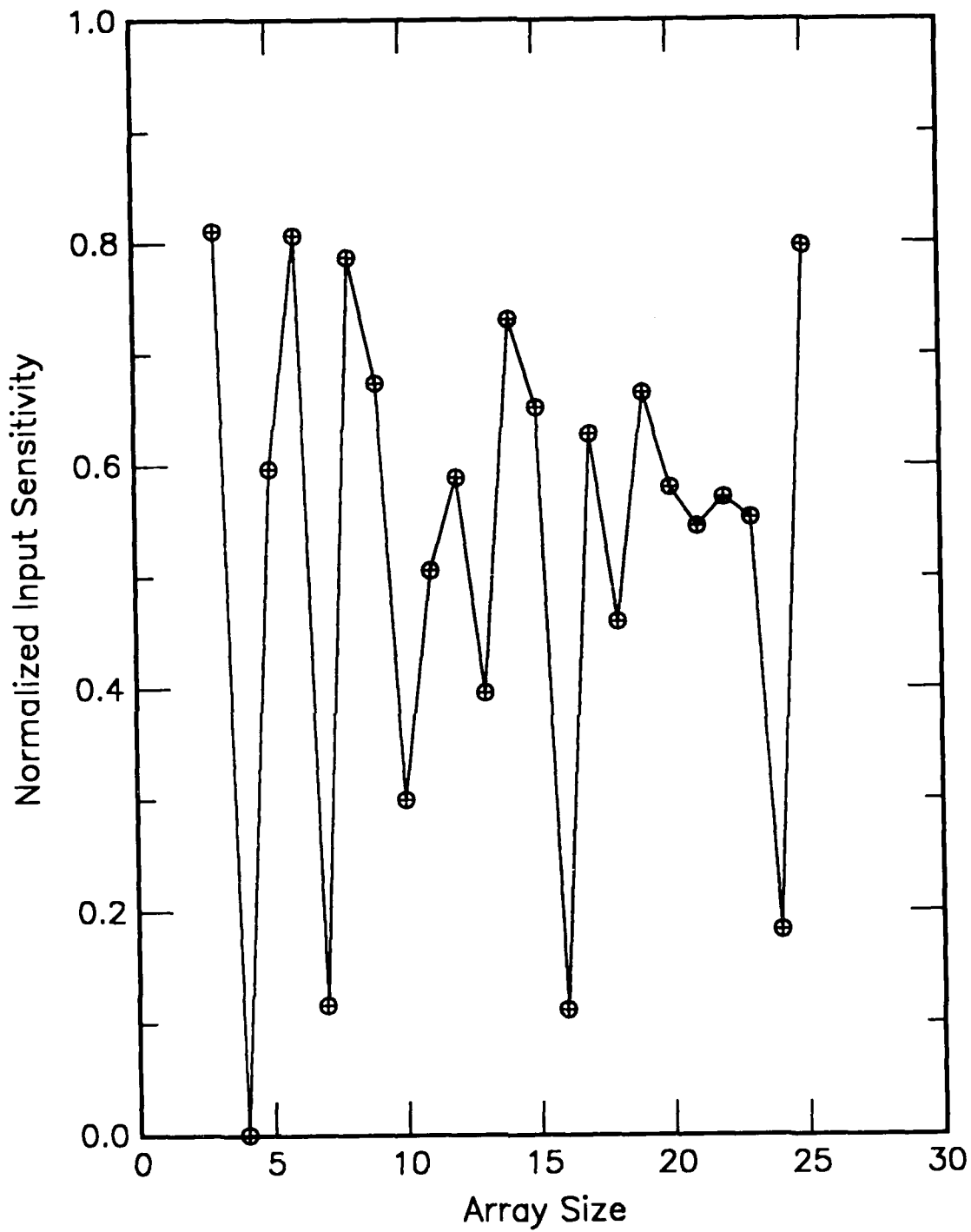


Figure E17. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 27

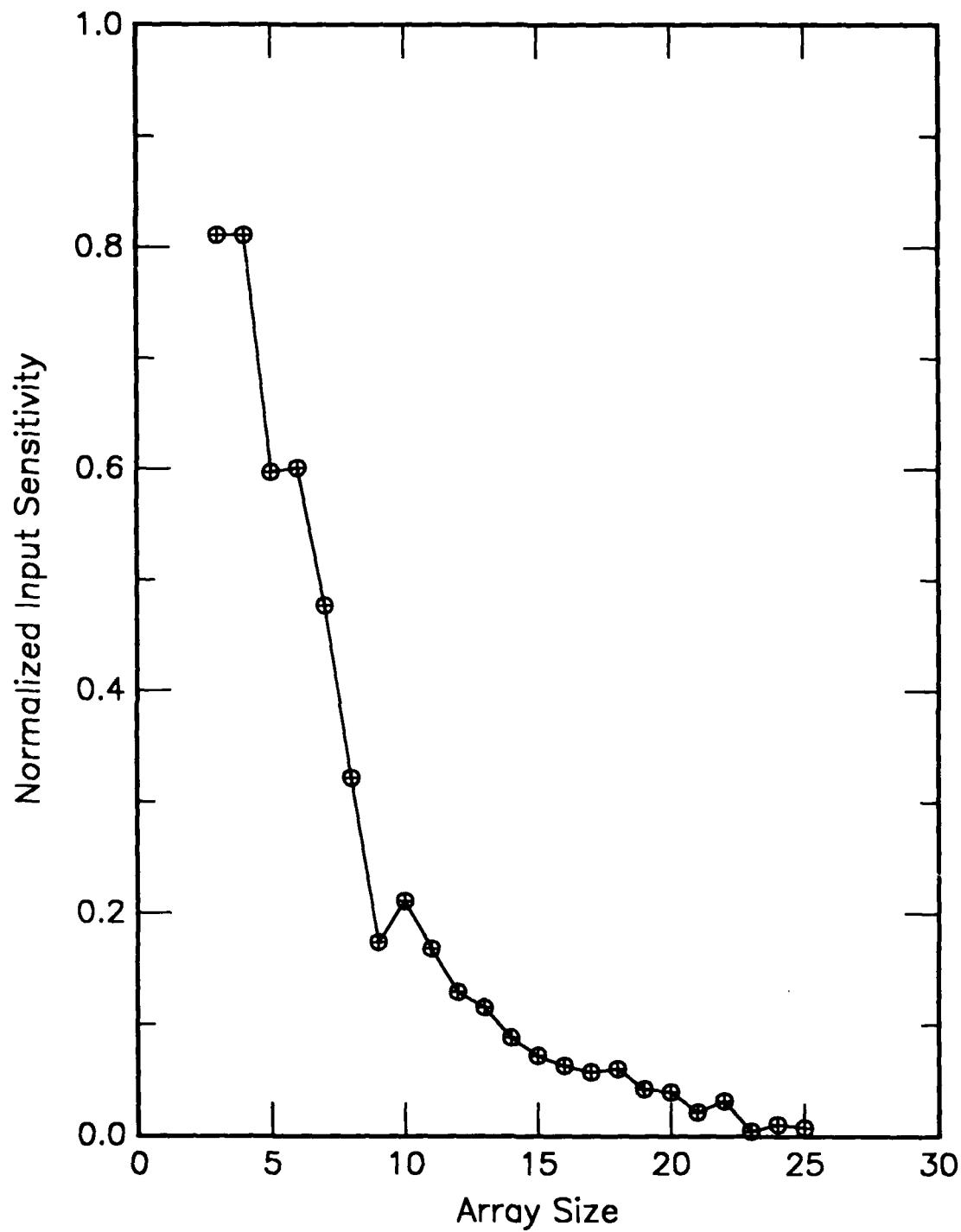


Figure E18. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

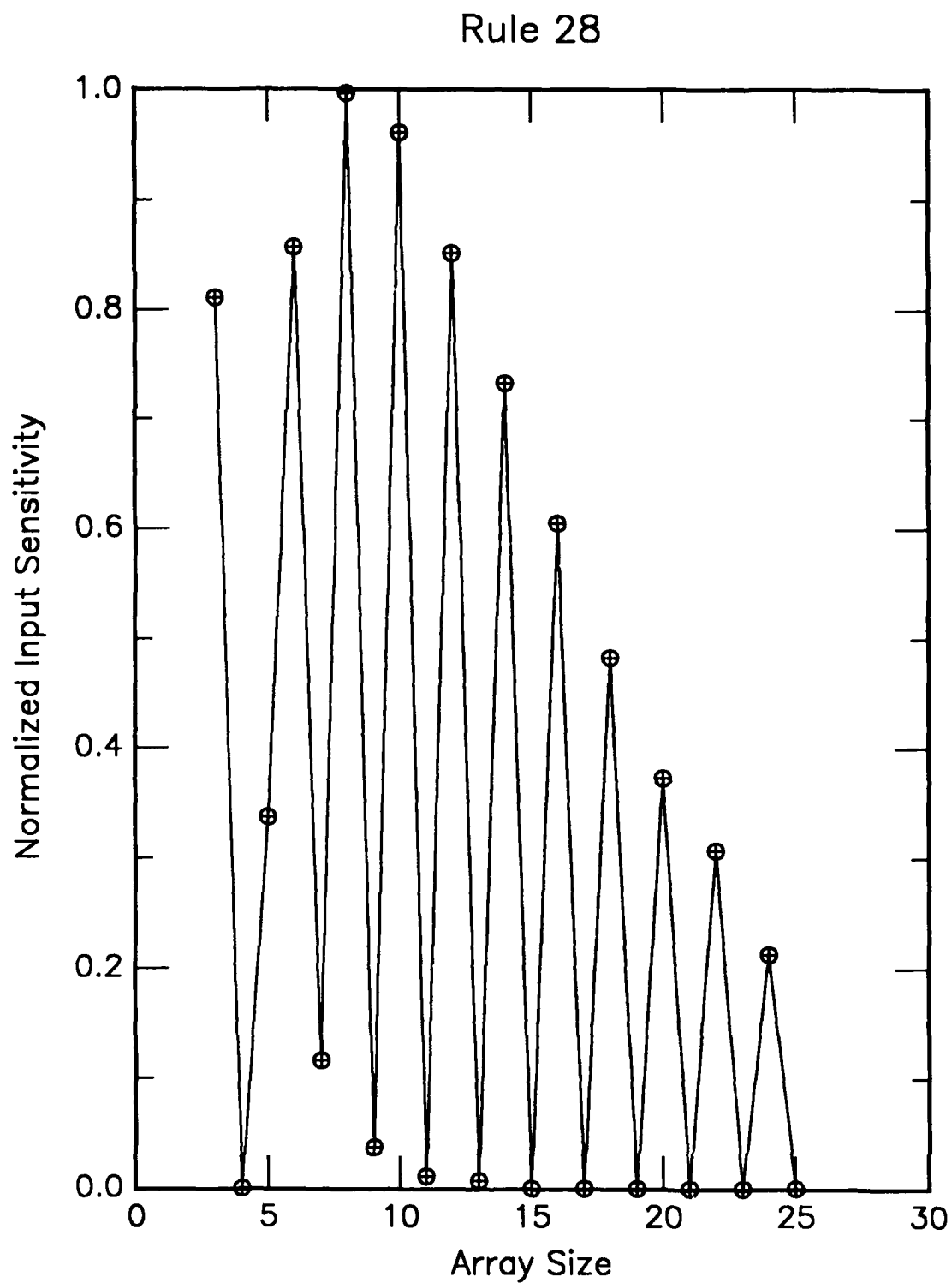


Figure E19. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 29

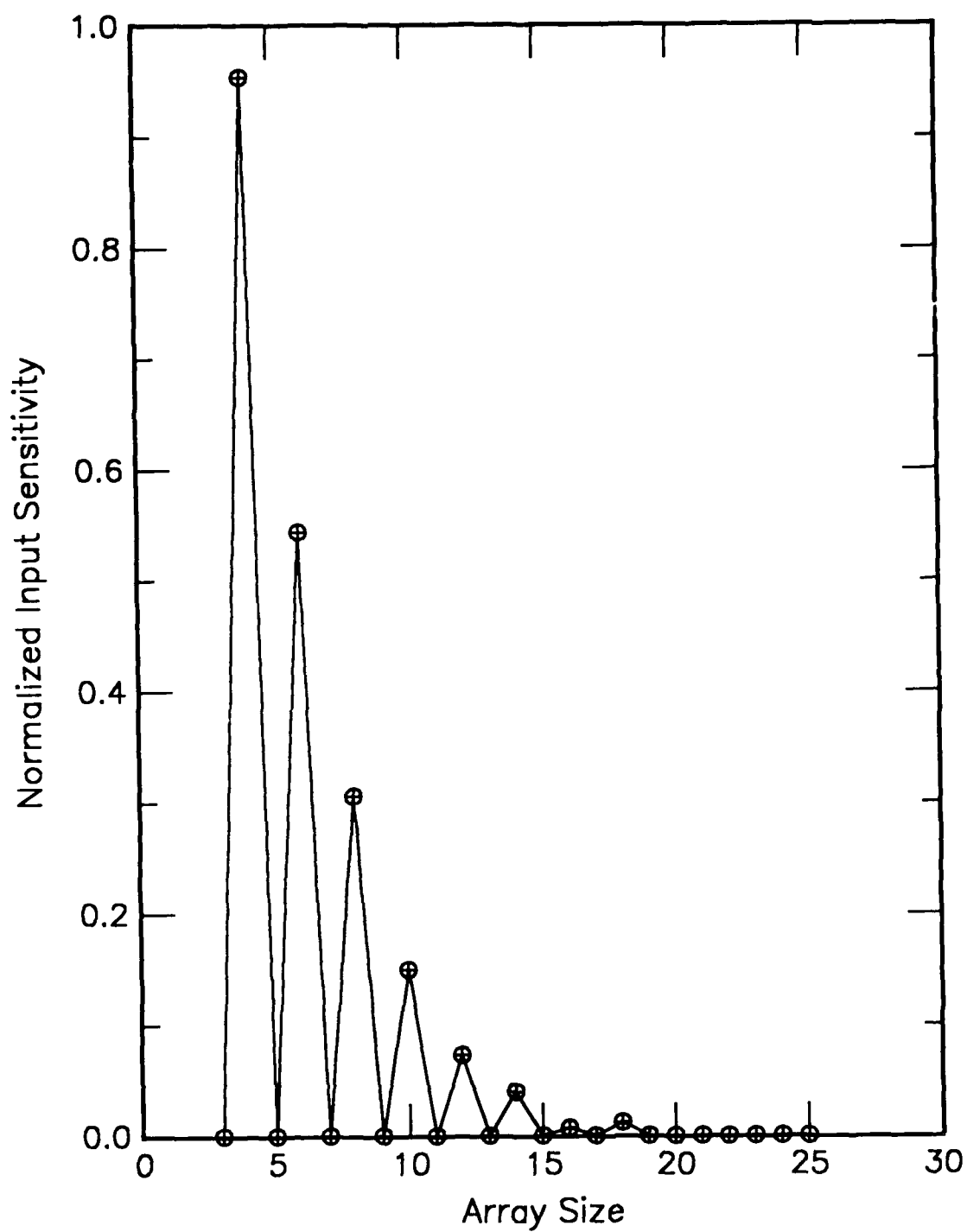


Figure E20. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 30

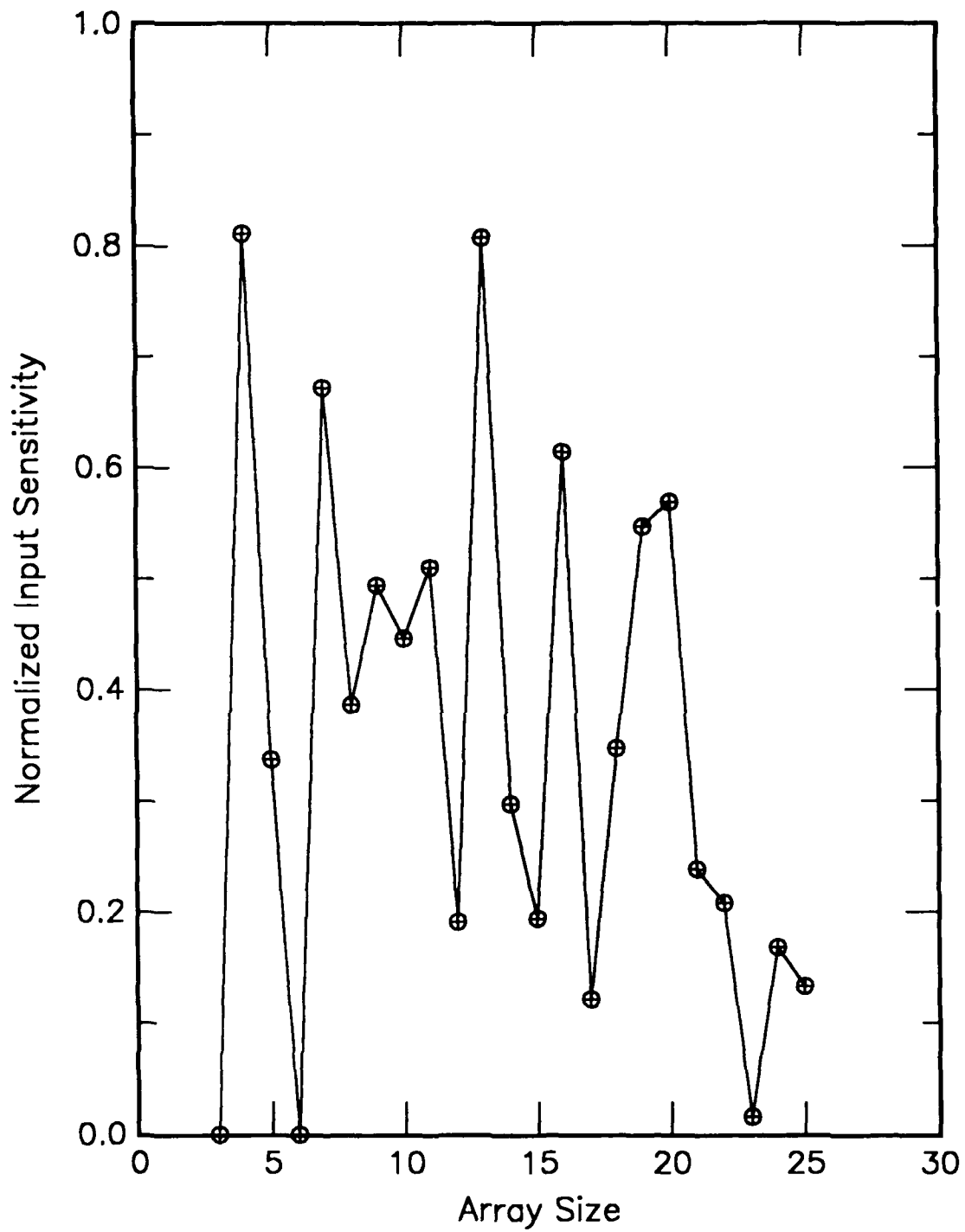


Figure E21. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 32

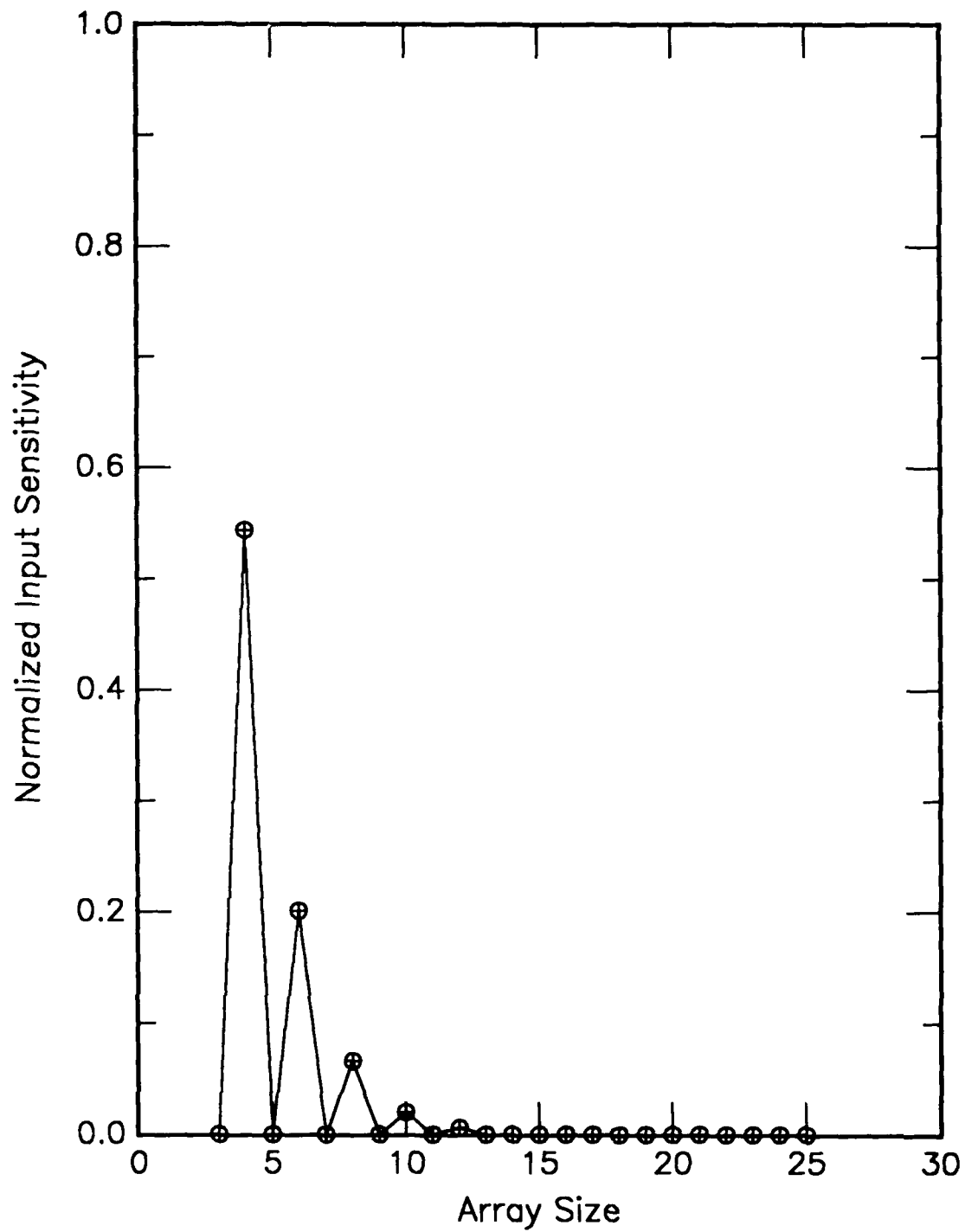


Figure E22. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 34

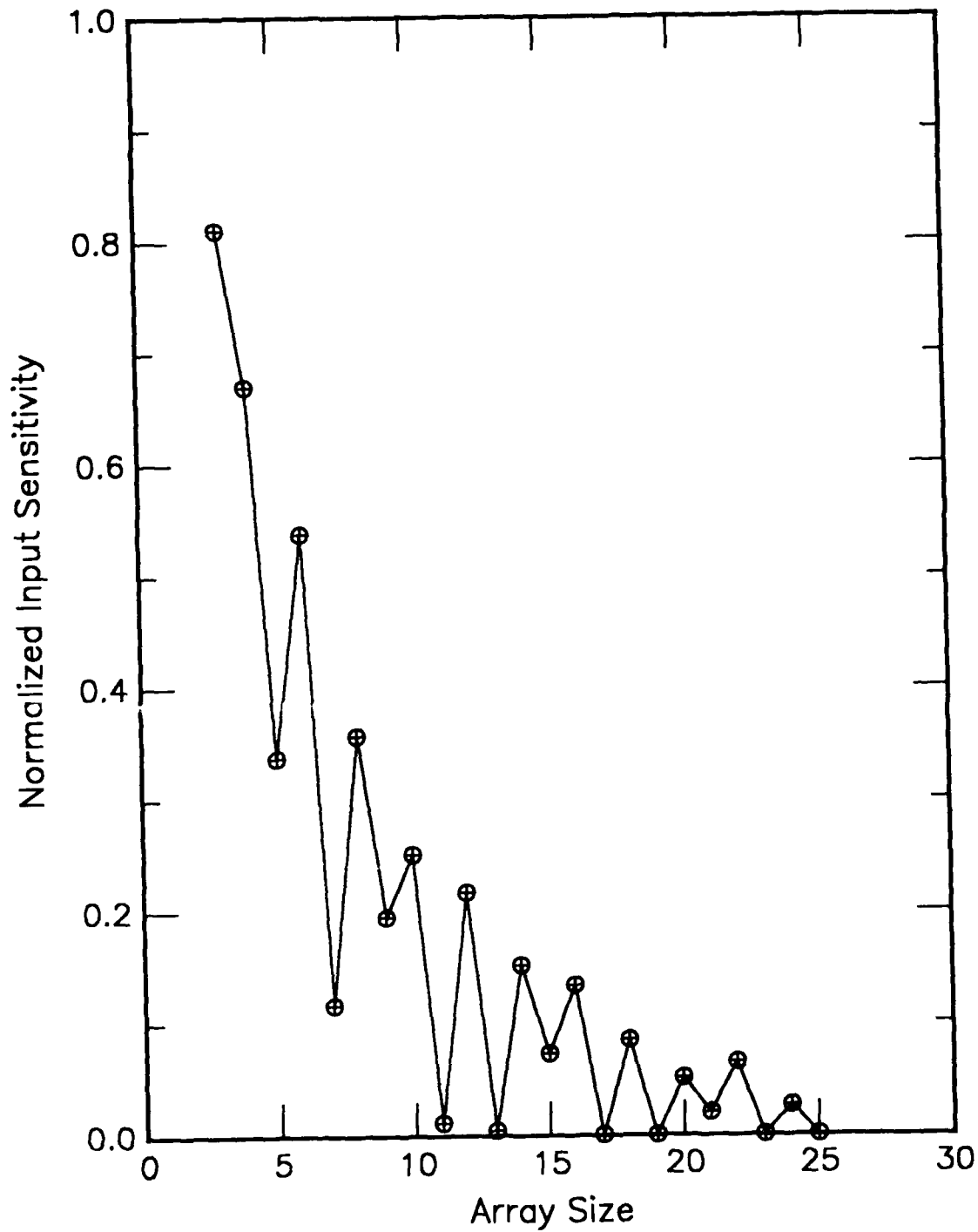


Figure E23. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 35

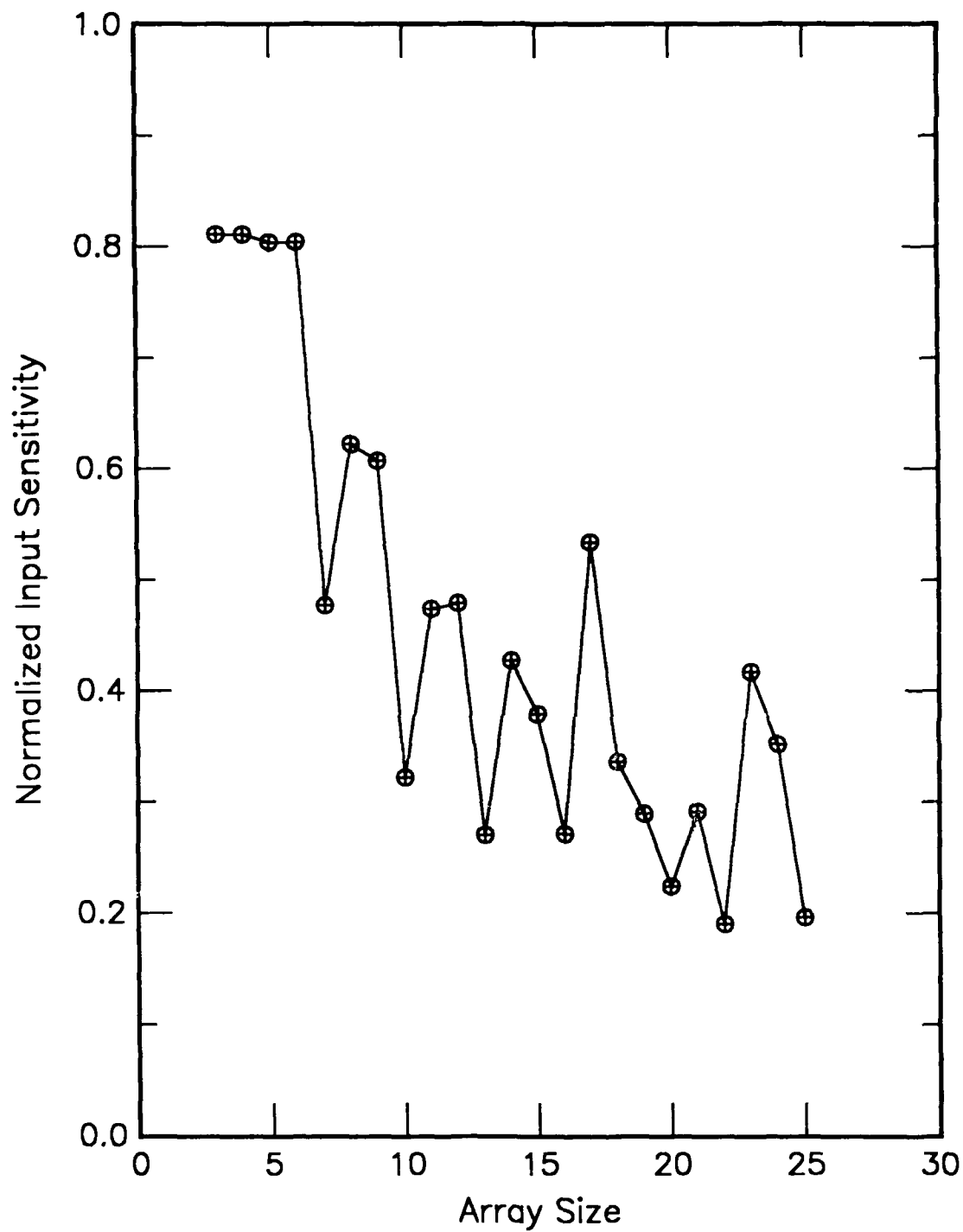


Figure E24. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 37

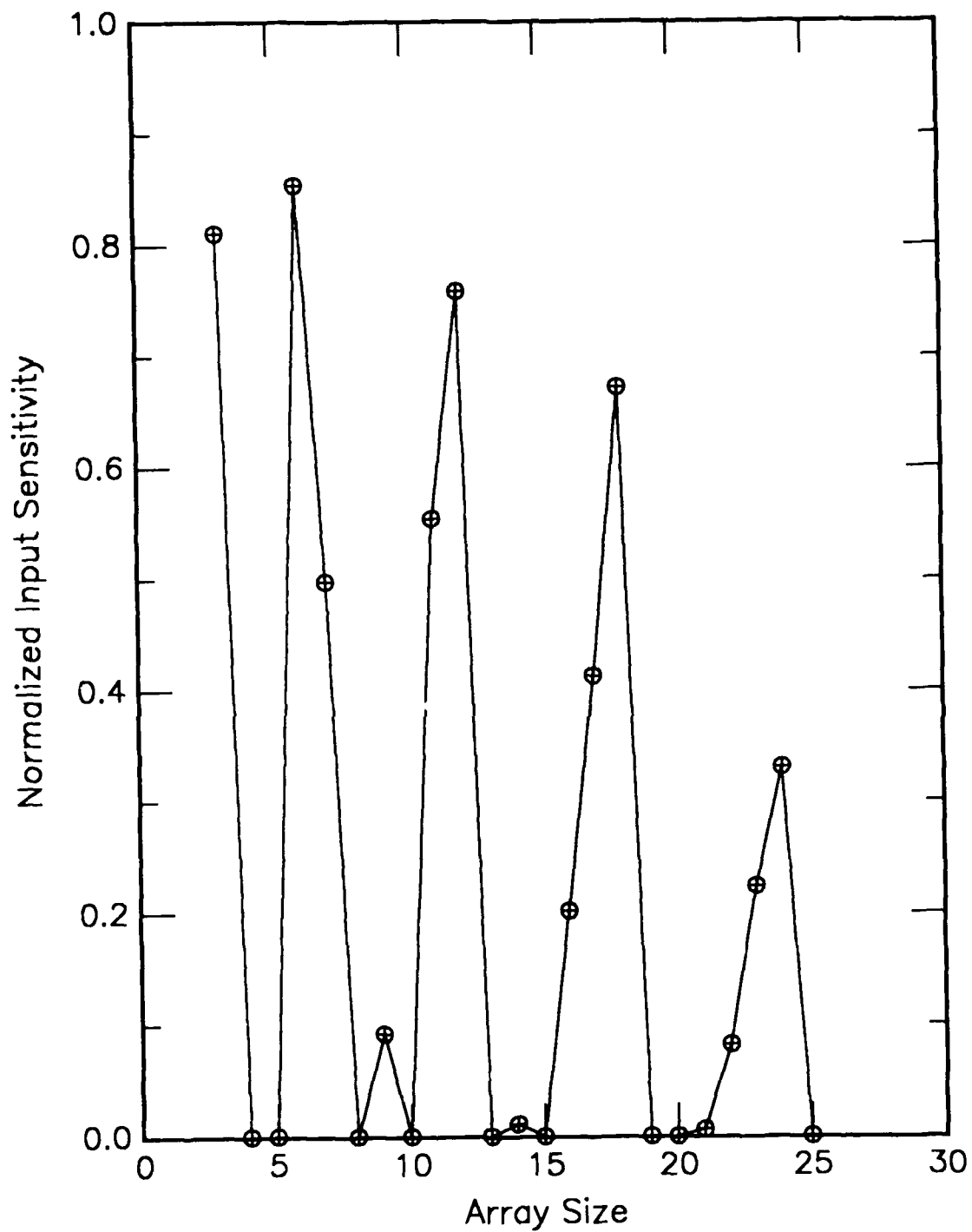


Figure E25. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 38

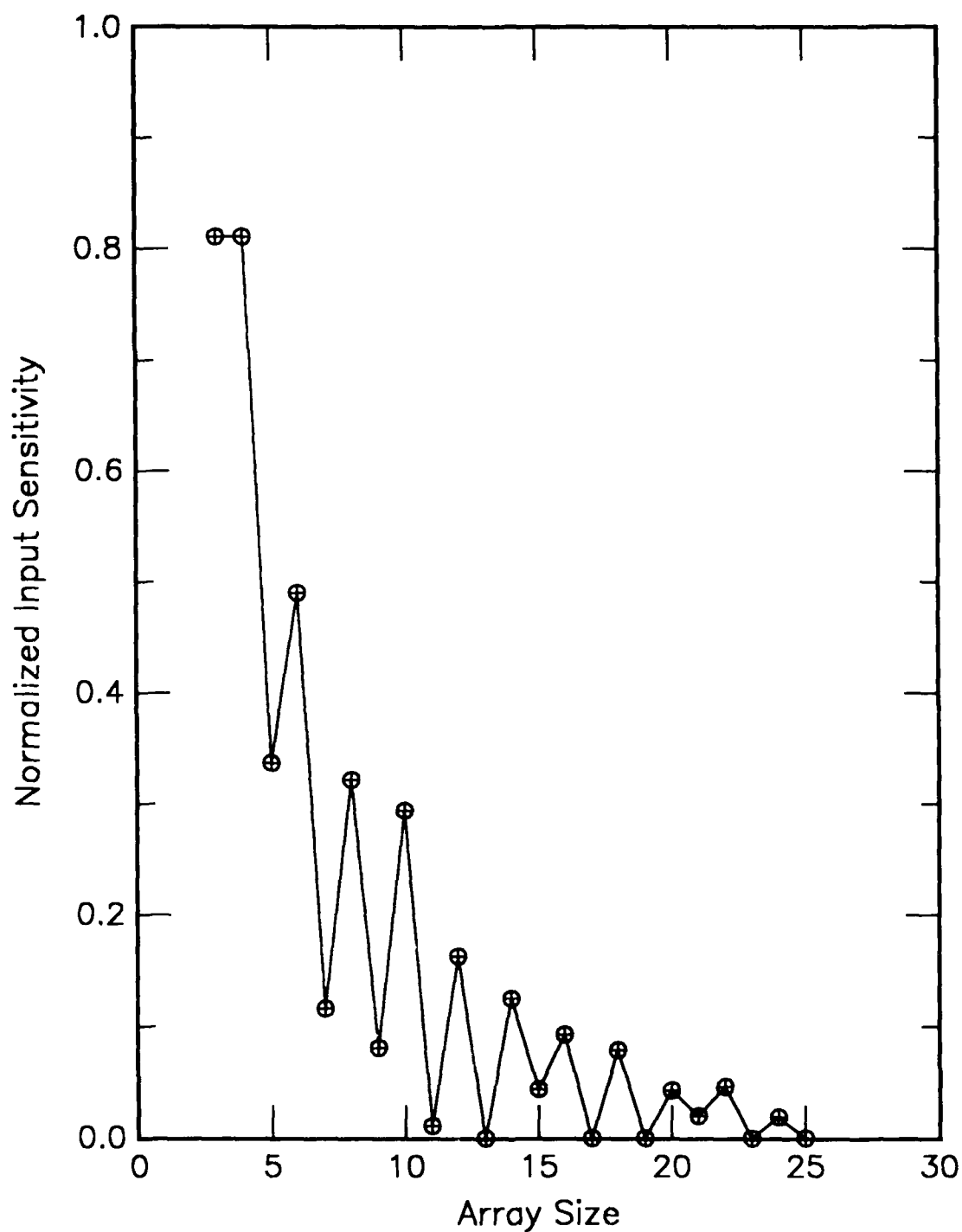


Figure E26. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

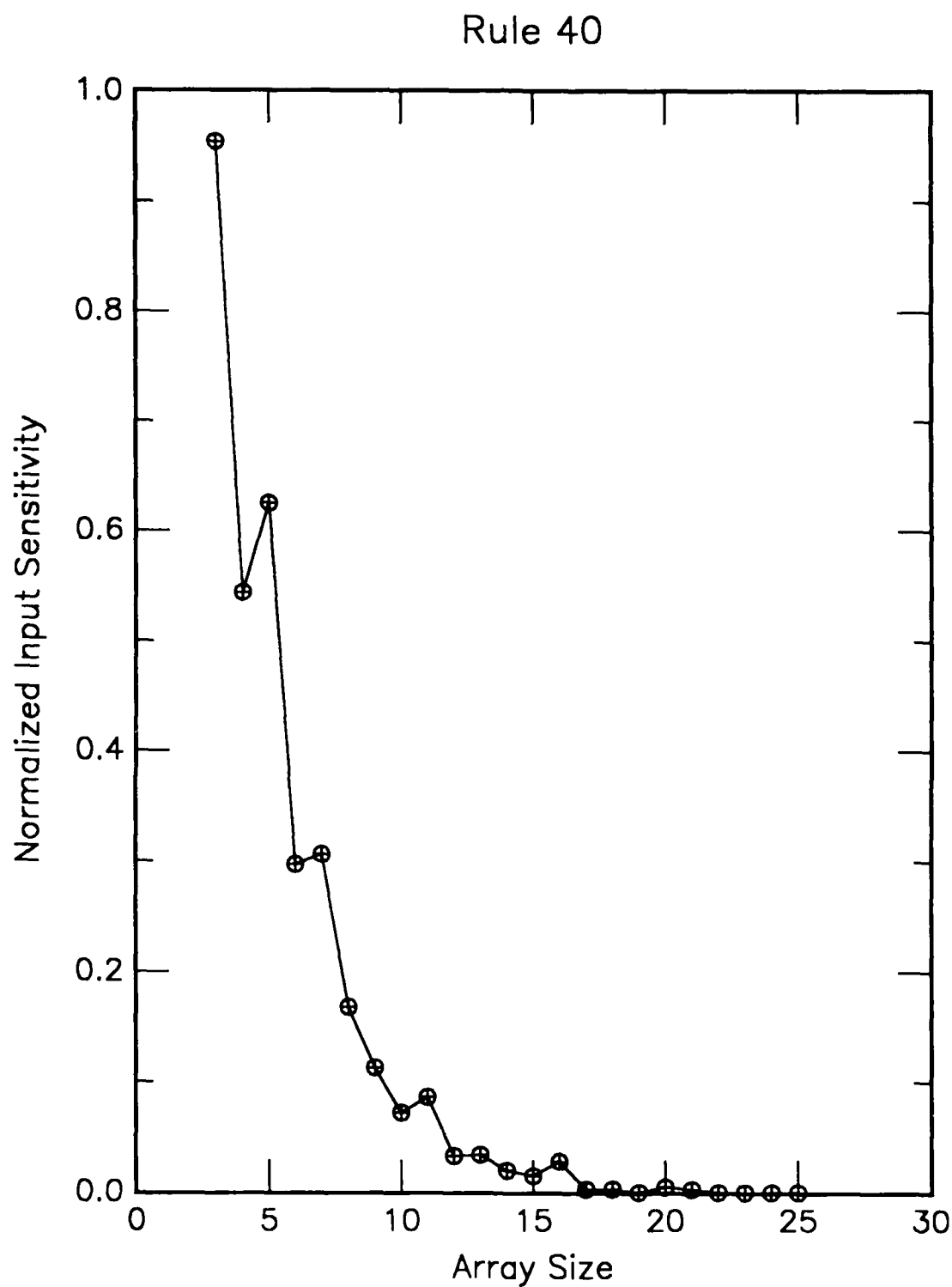


Figure E27. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 41

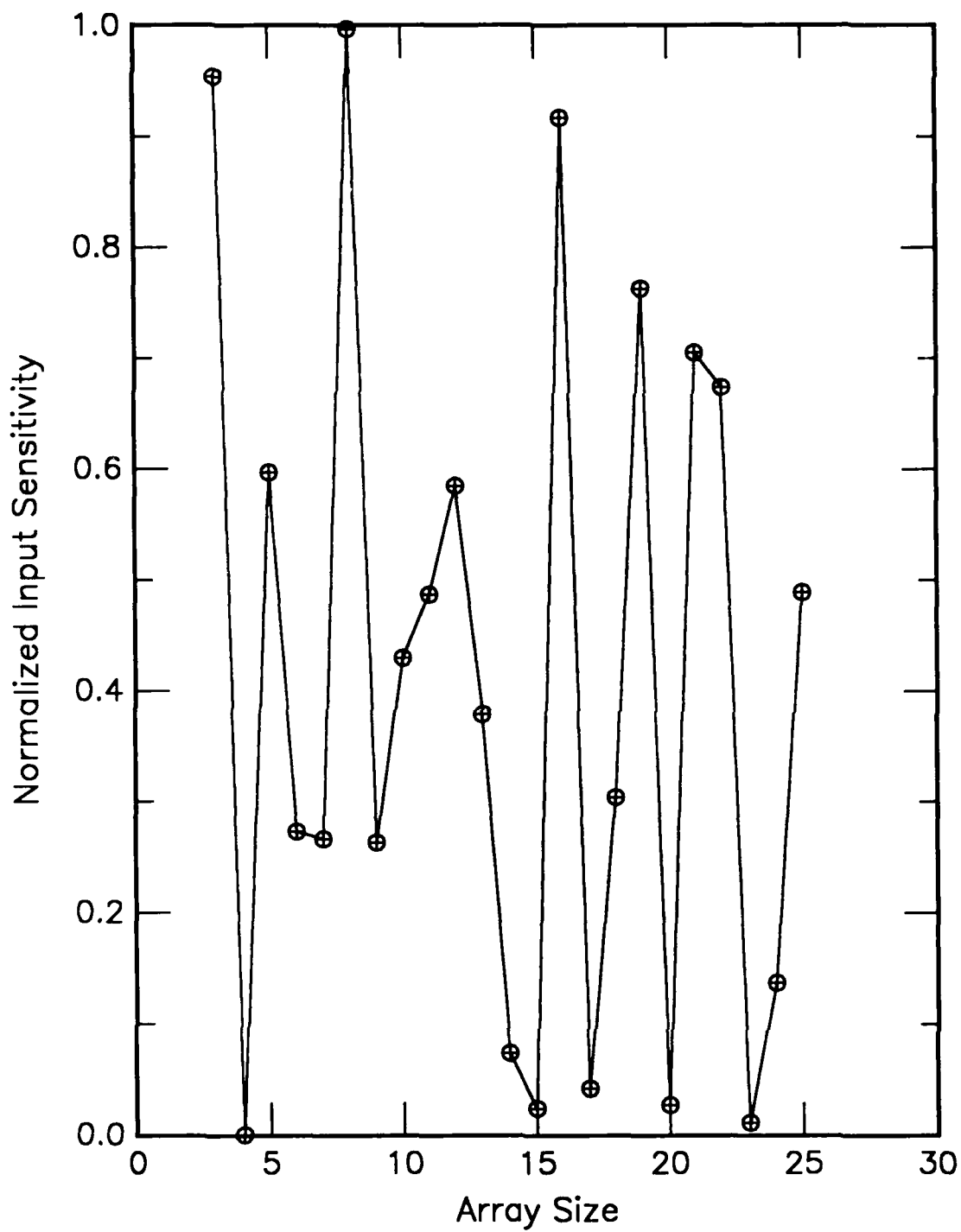


Figure E28. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 42

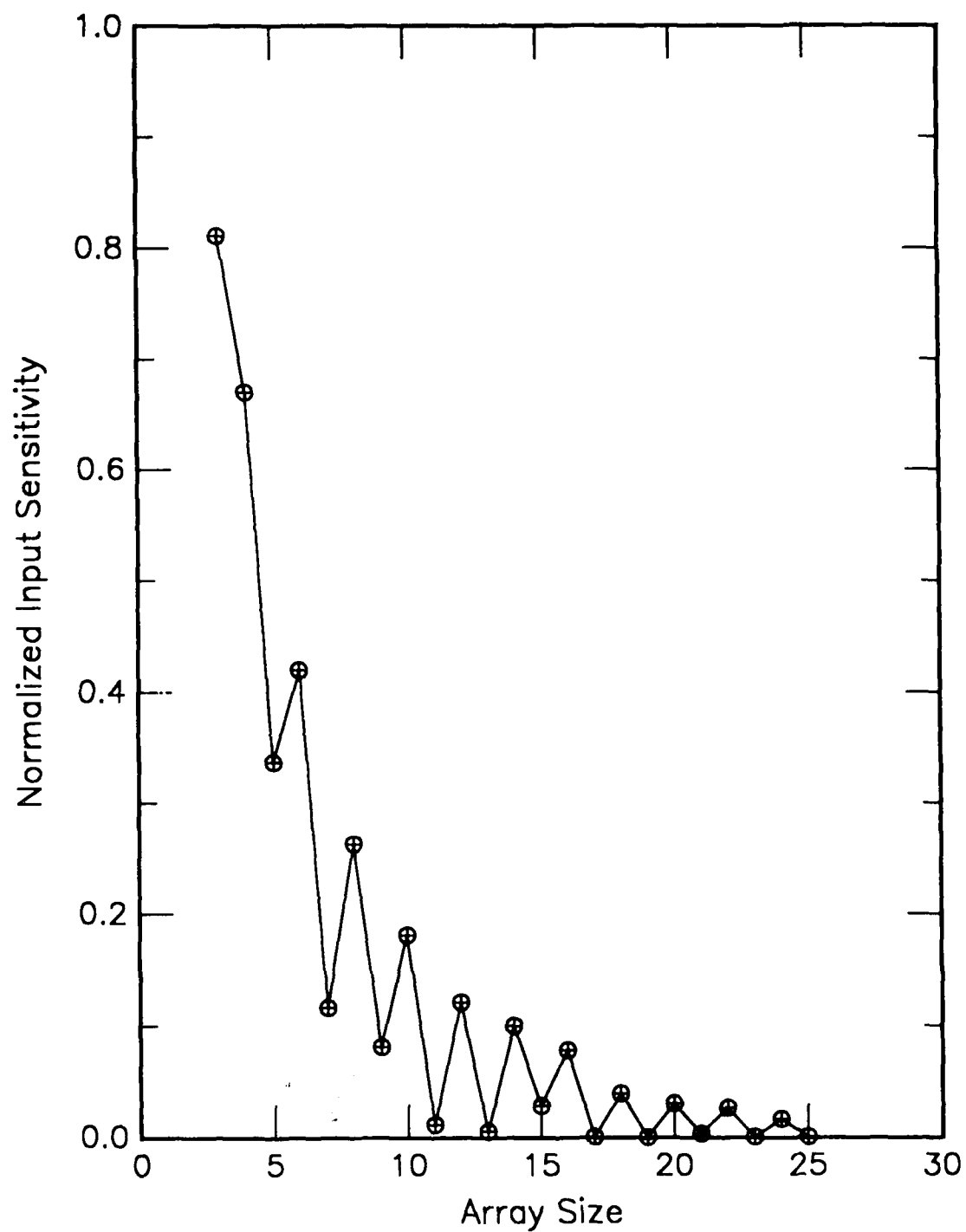


Figure E29. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

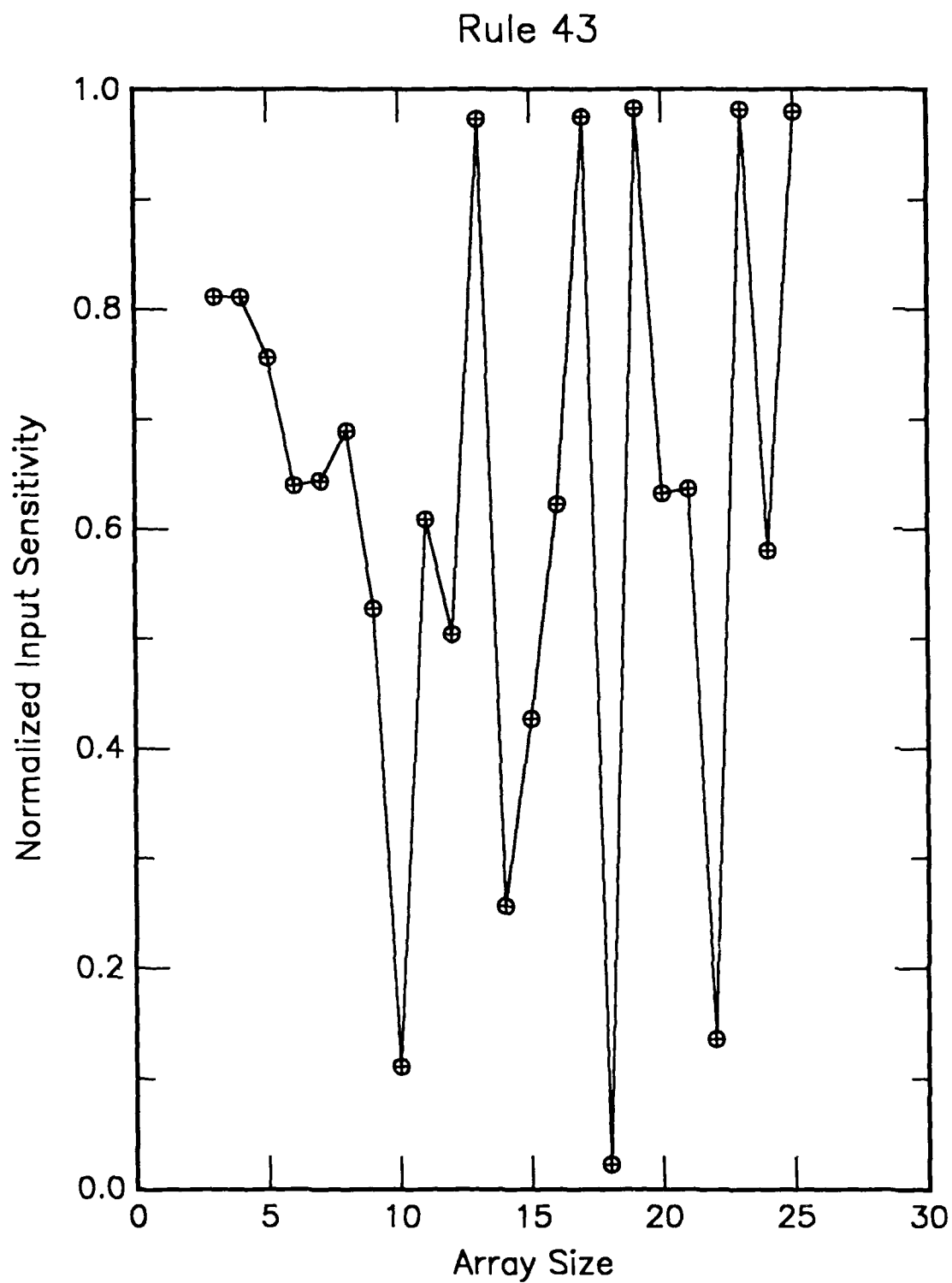


Figure E30. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 45

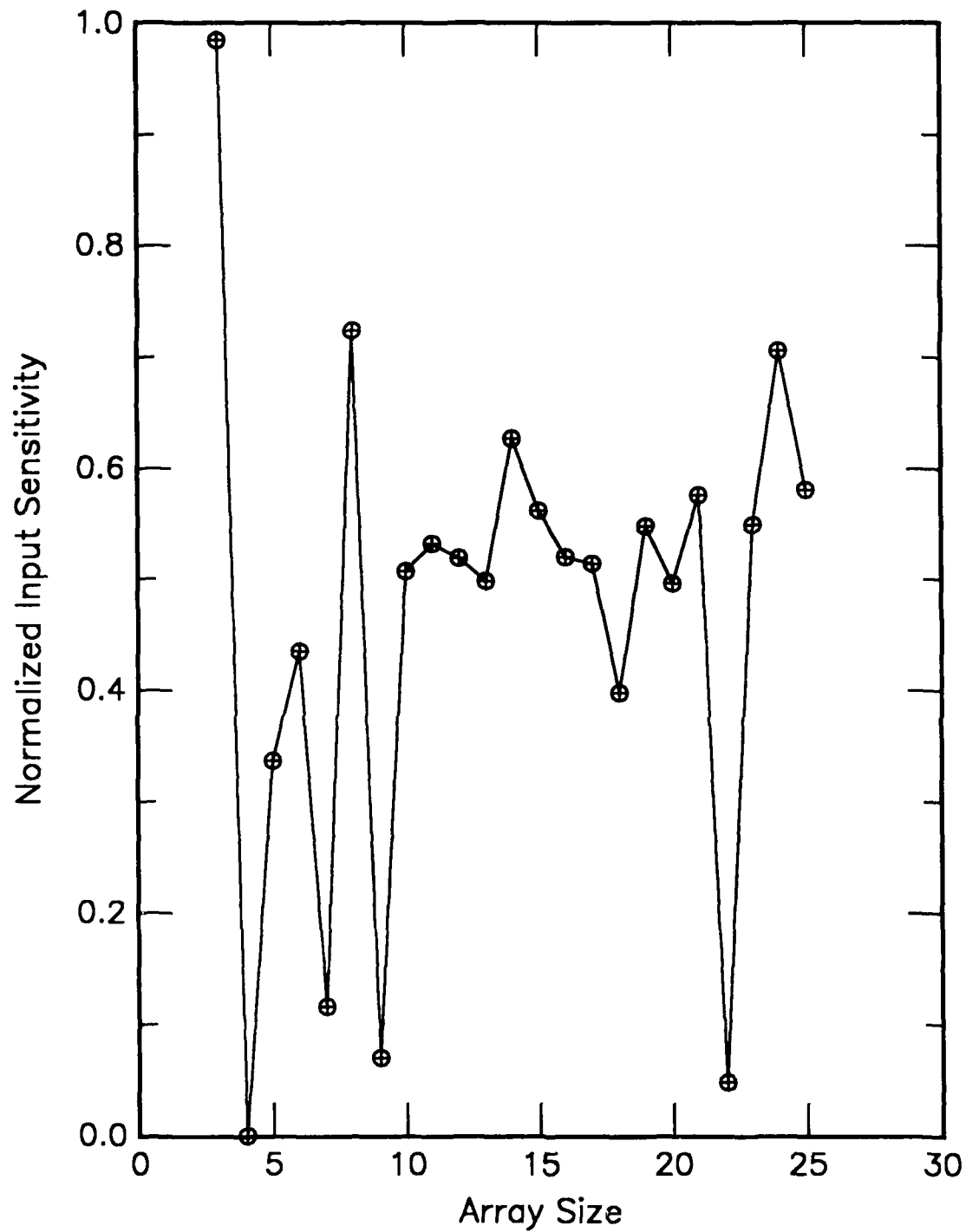


Figure E31. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 46

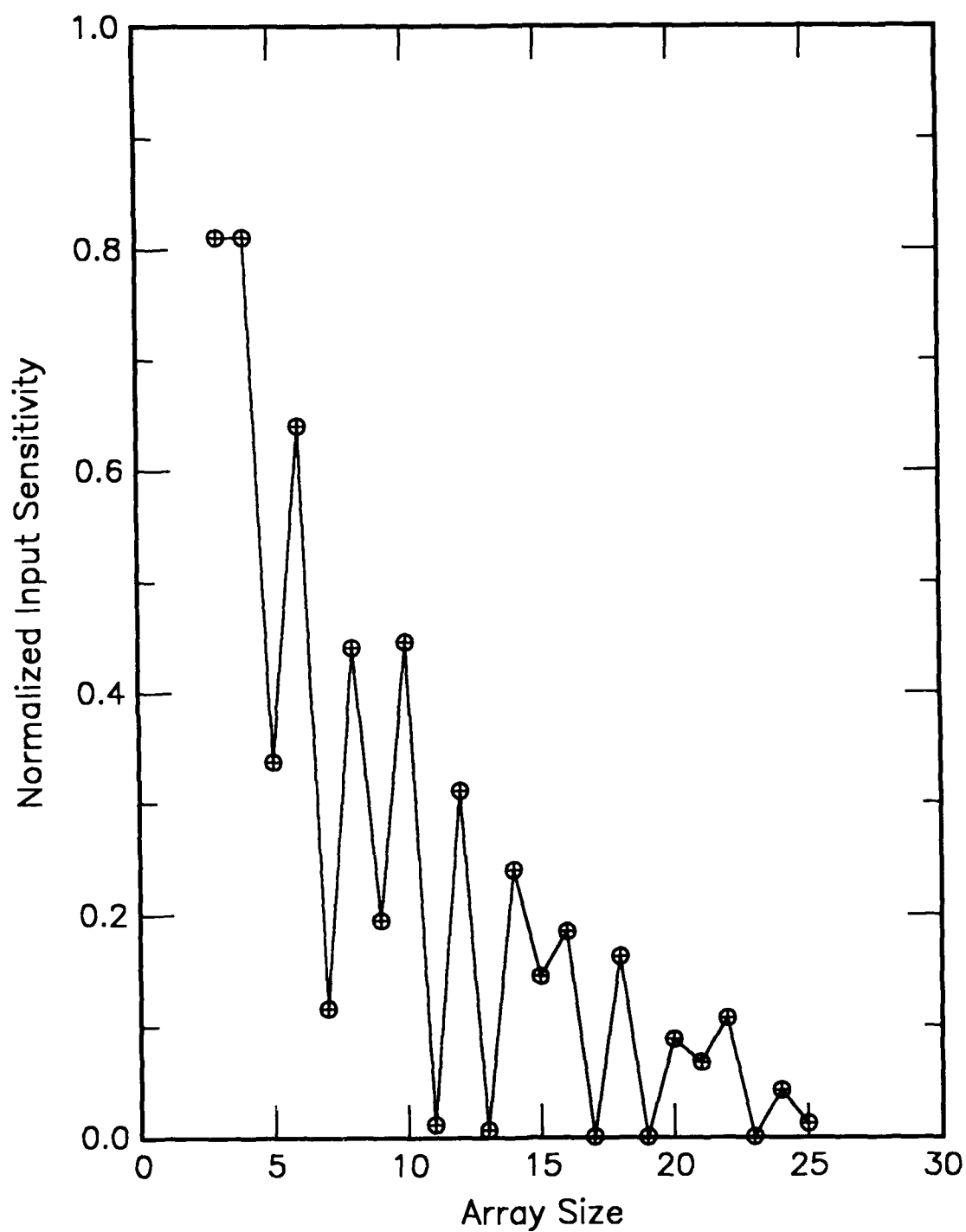


Figure E32. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 50

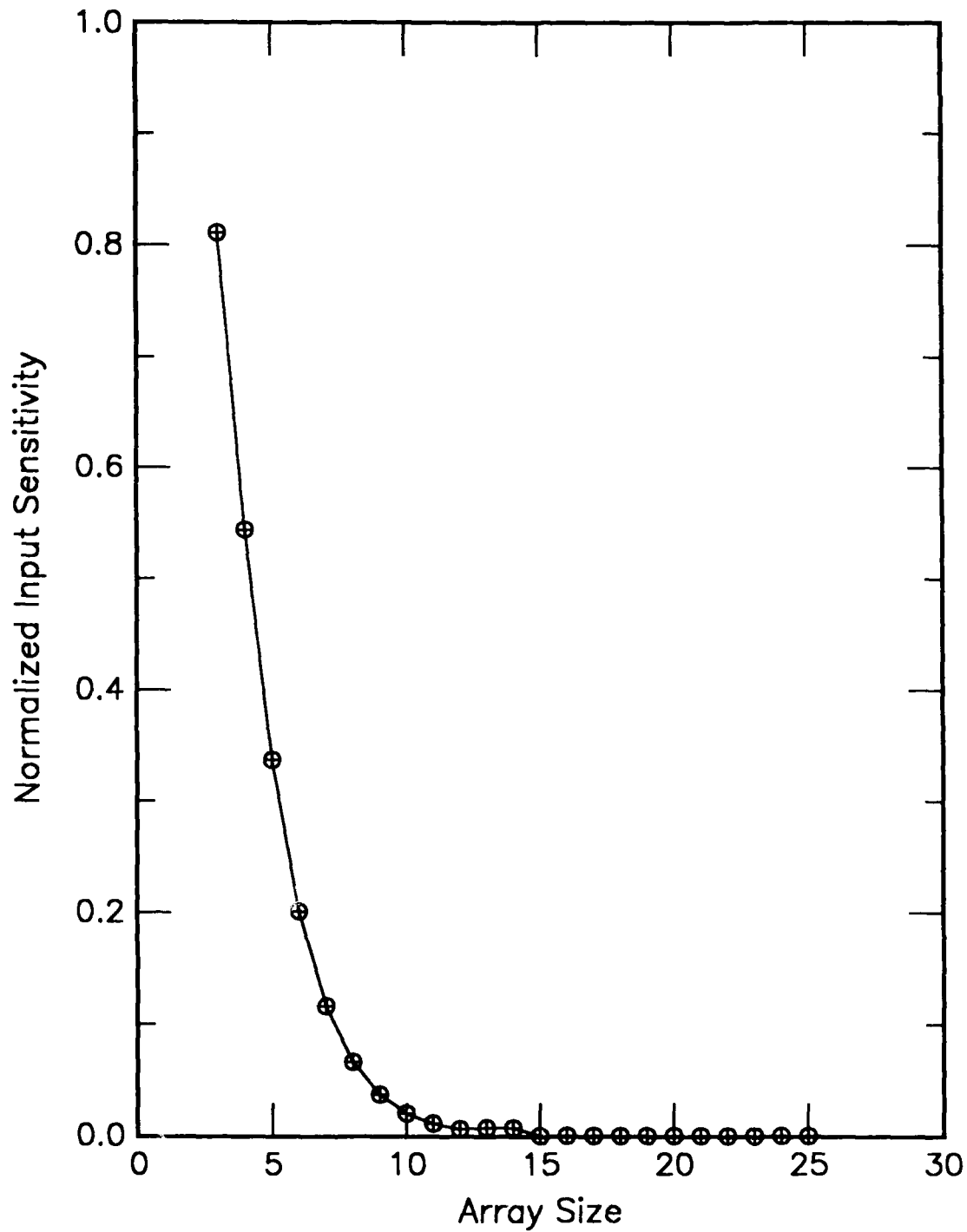


Figure E33. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 54

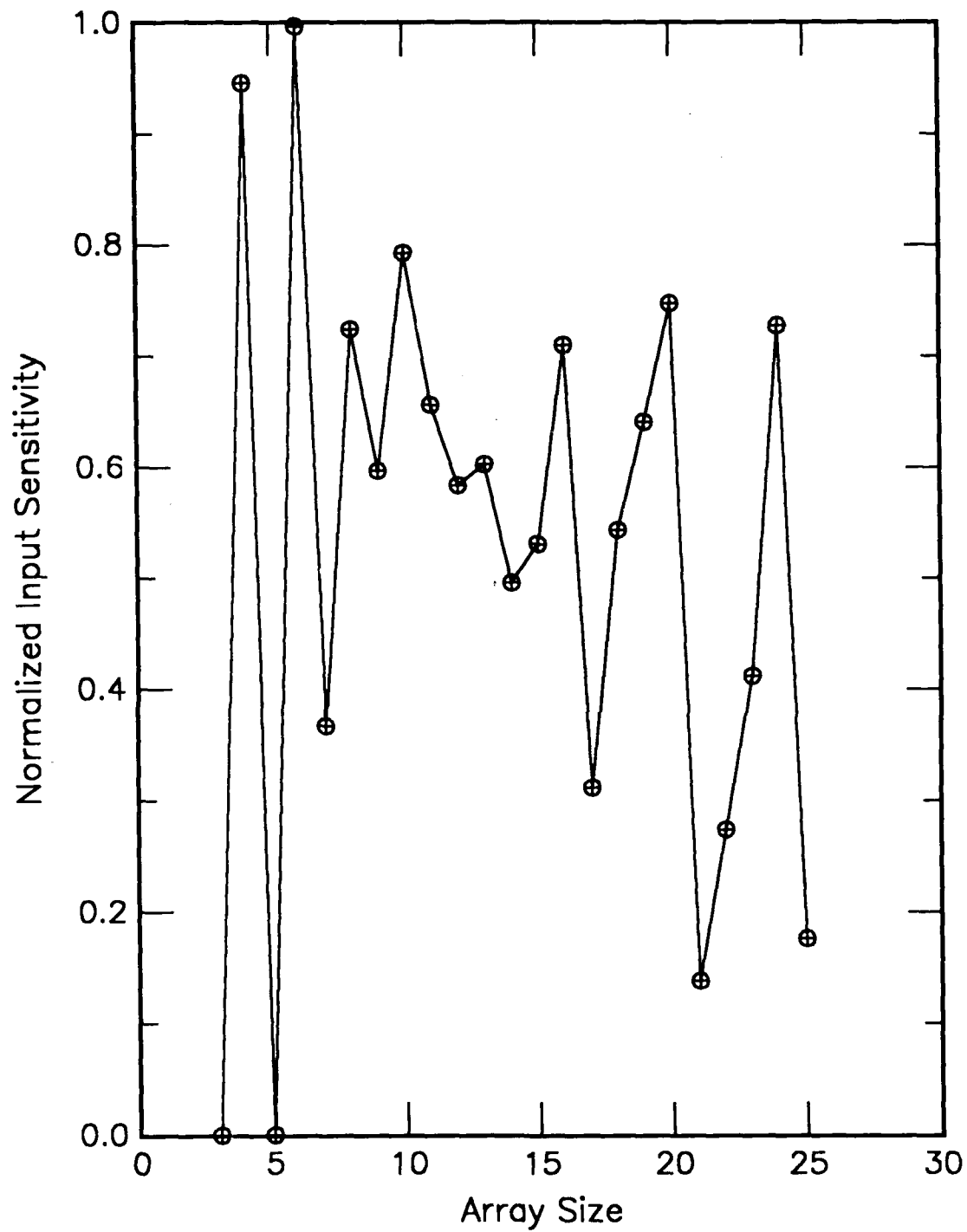


Figure E34. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 56

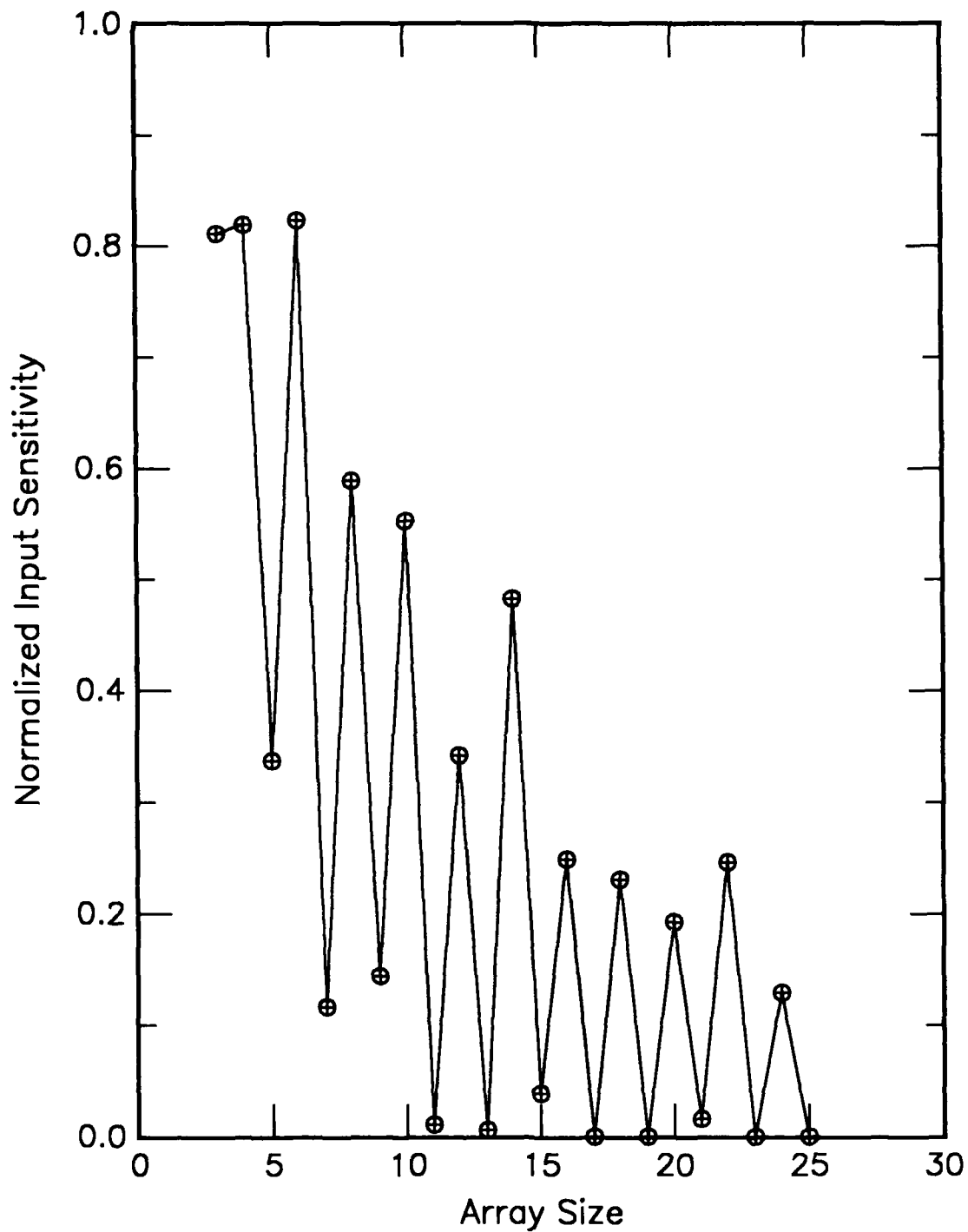


Figure E35. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 57

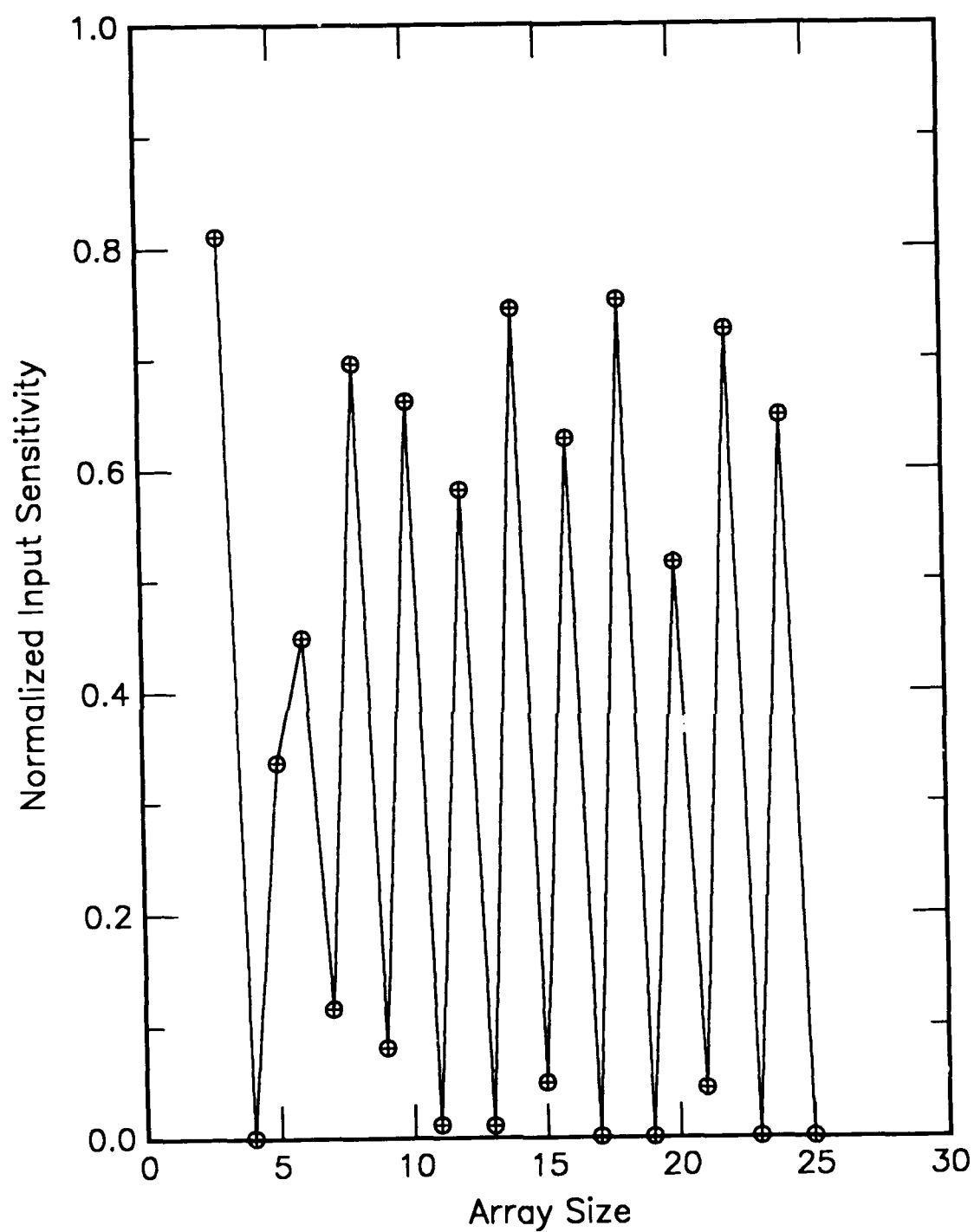


Figure E36. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

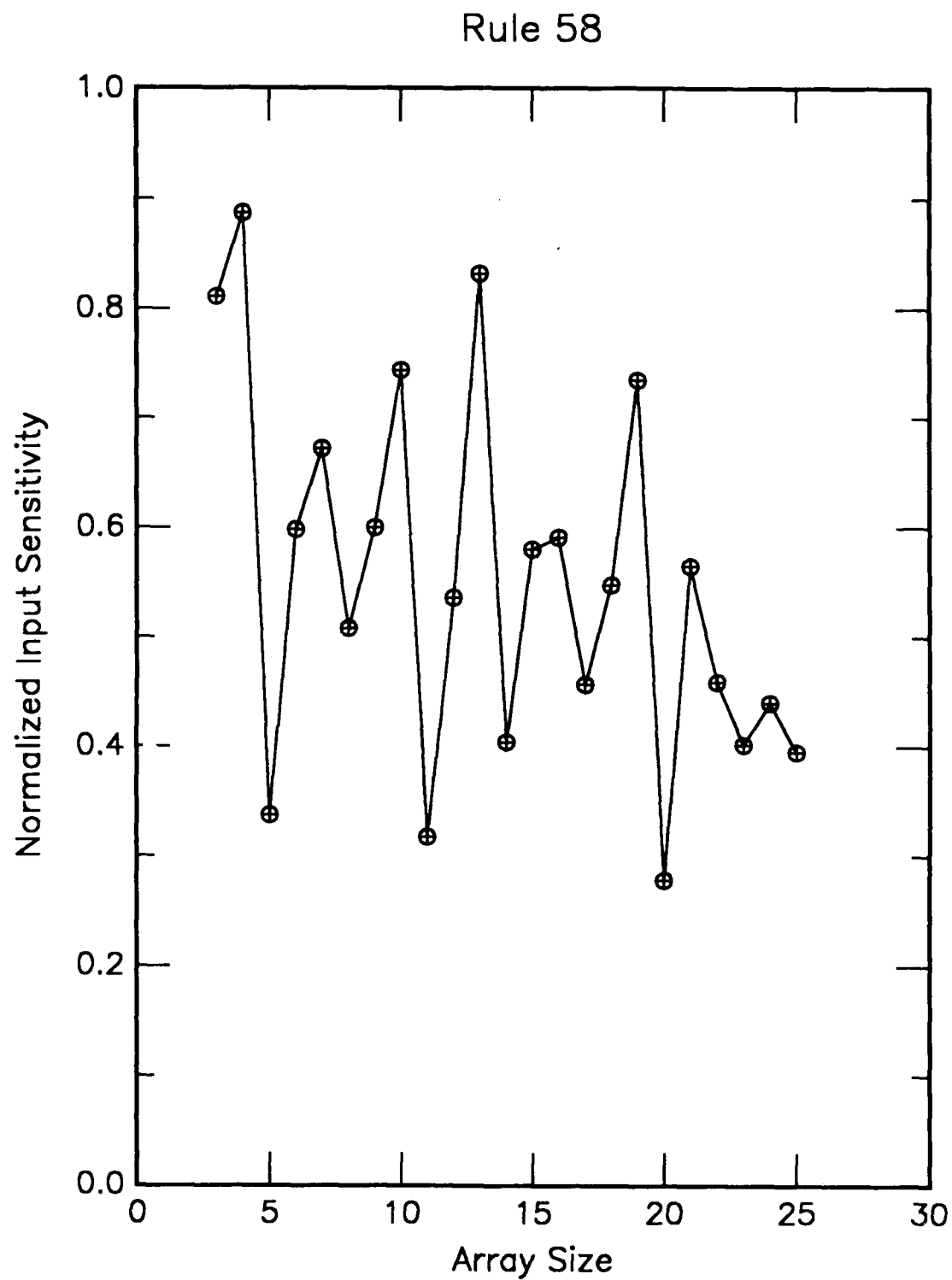


Figure E37. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

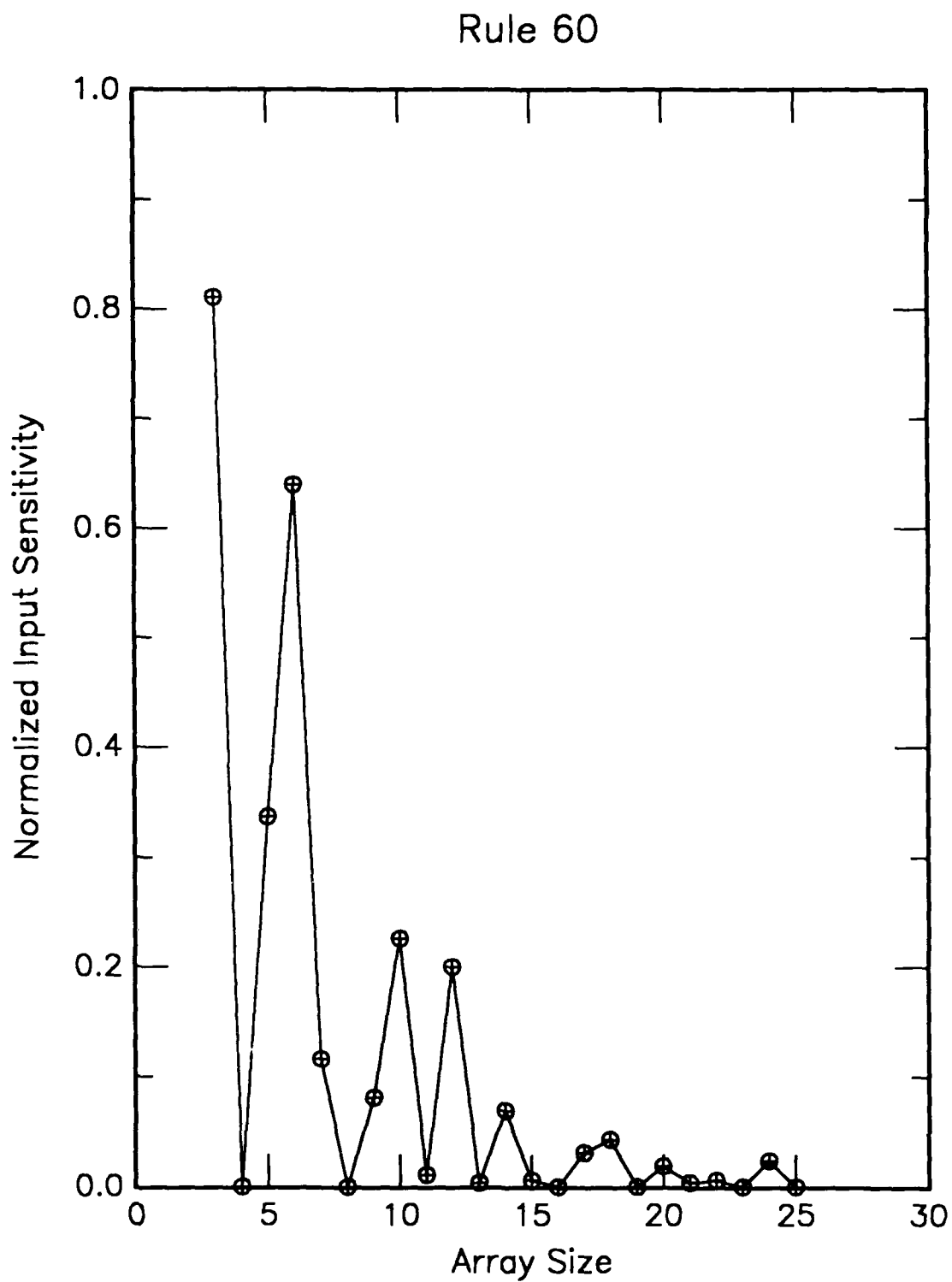


Figure E38. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 62

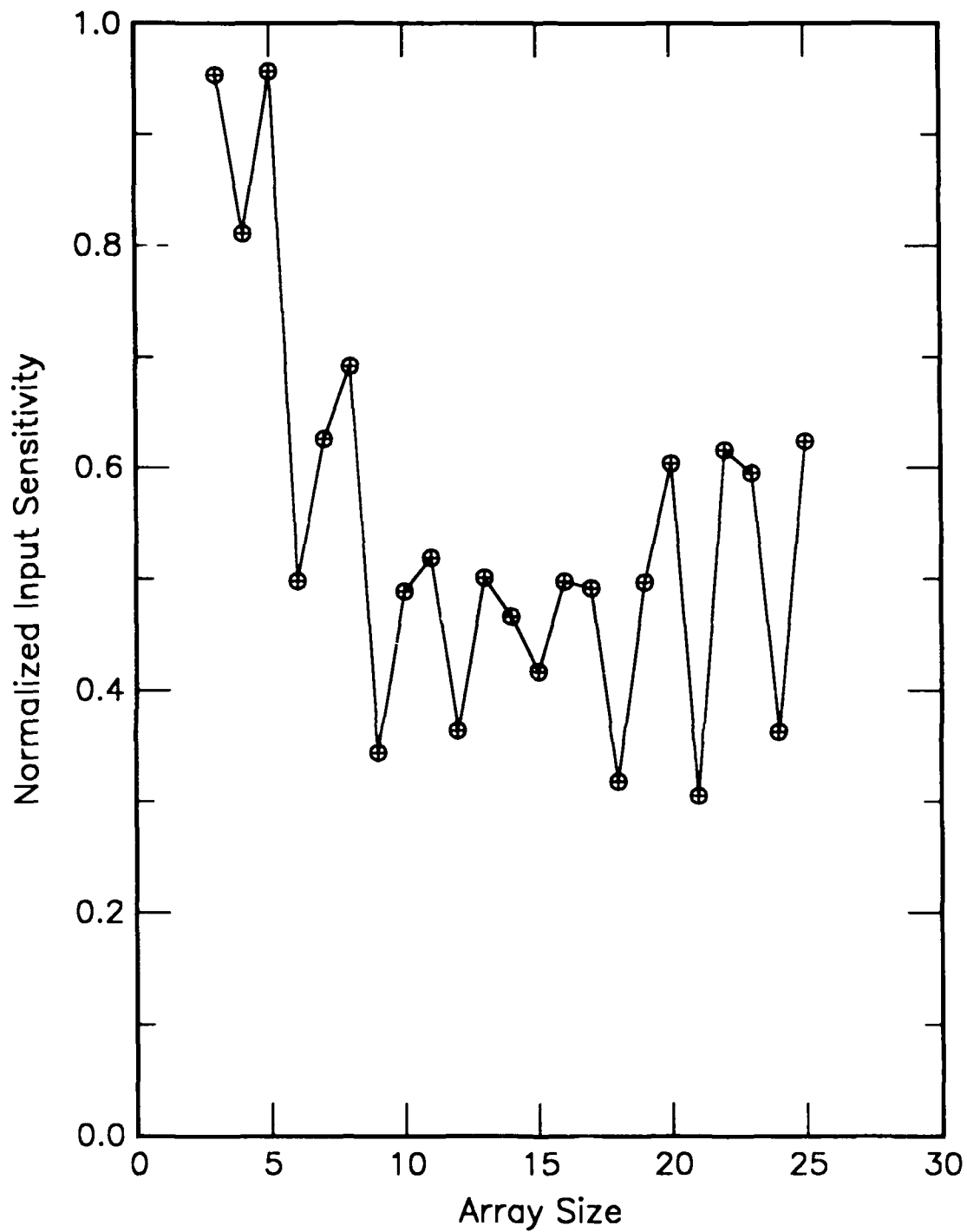


Figure E39. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 73

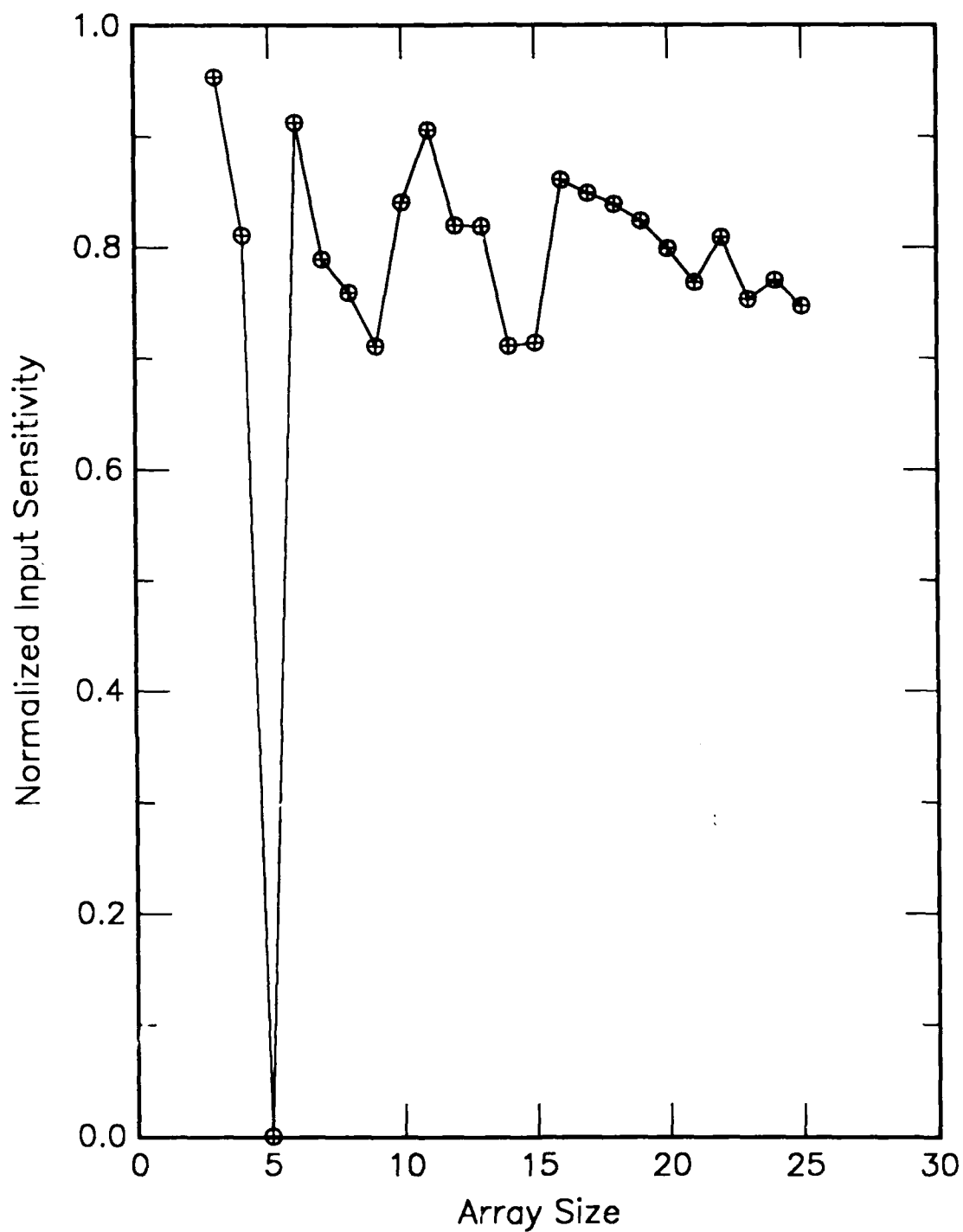


Figure E40. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 74

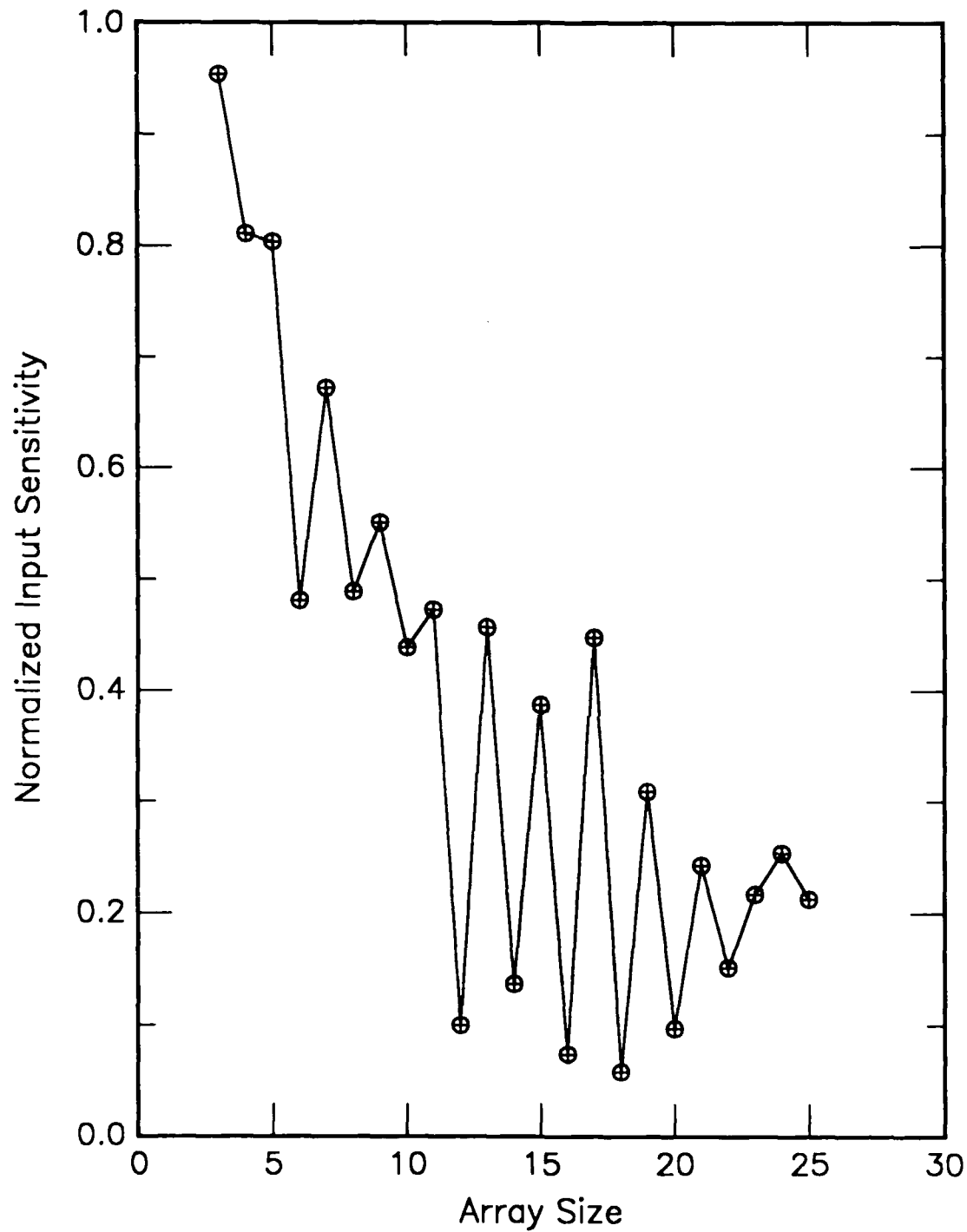


Figure E41. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 77

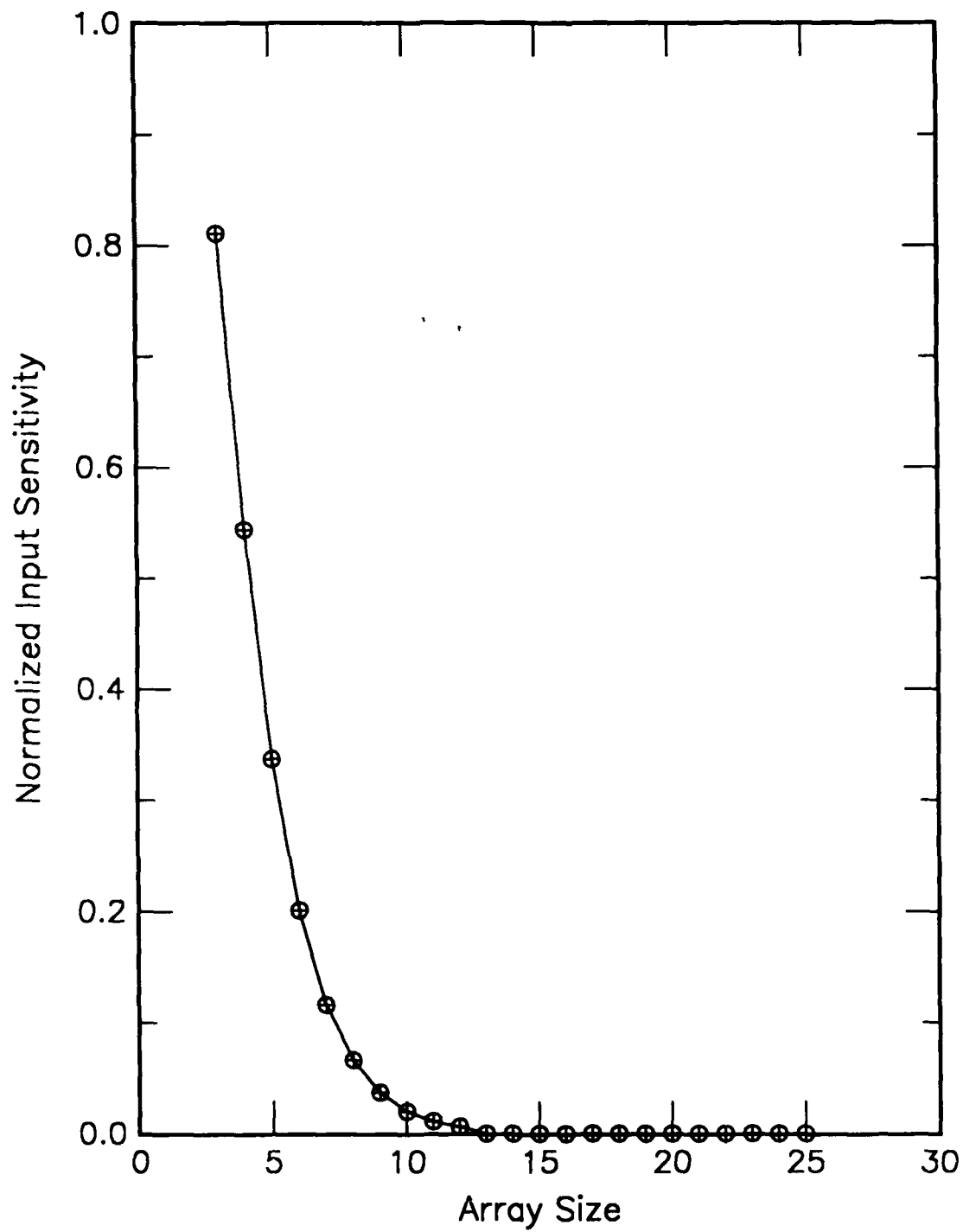


Figure E42. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 90

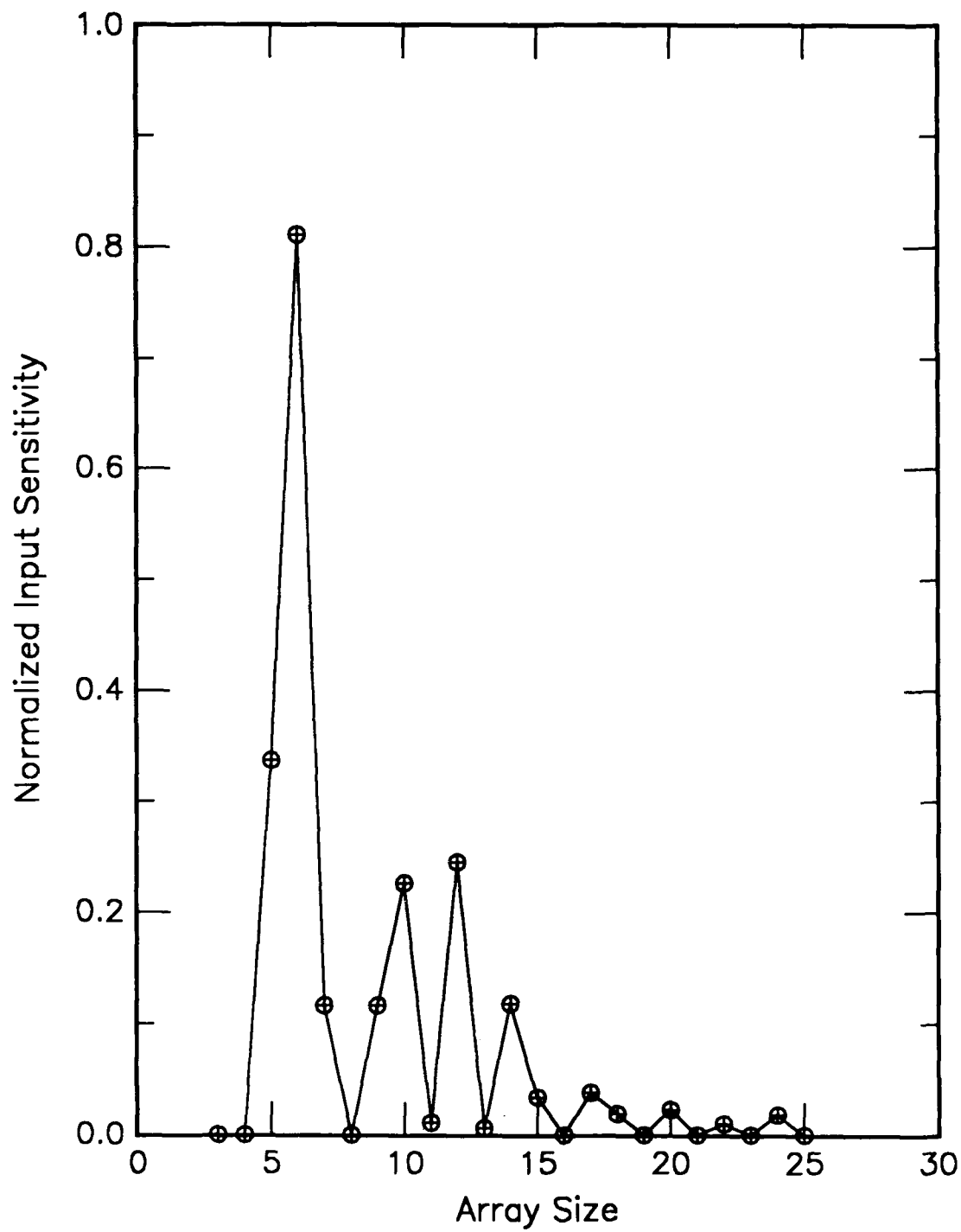


Figure E43. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 94

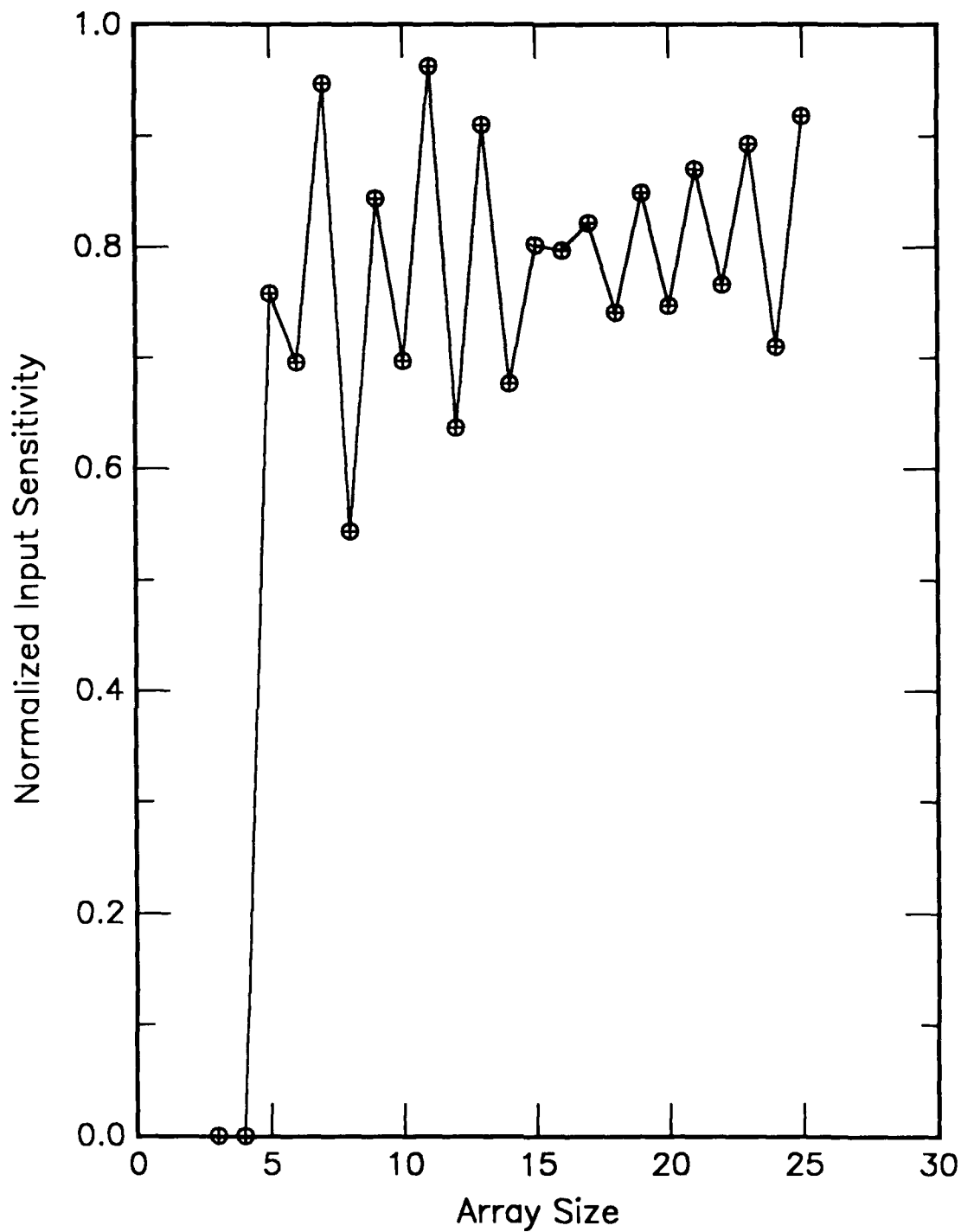


Figure E44. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 104

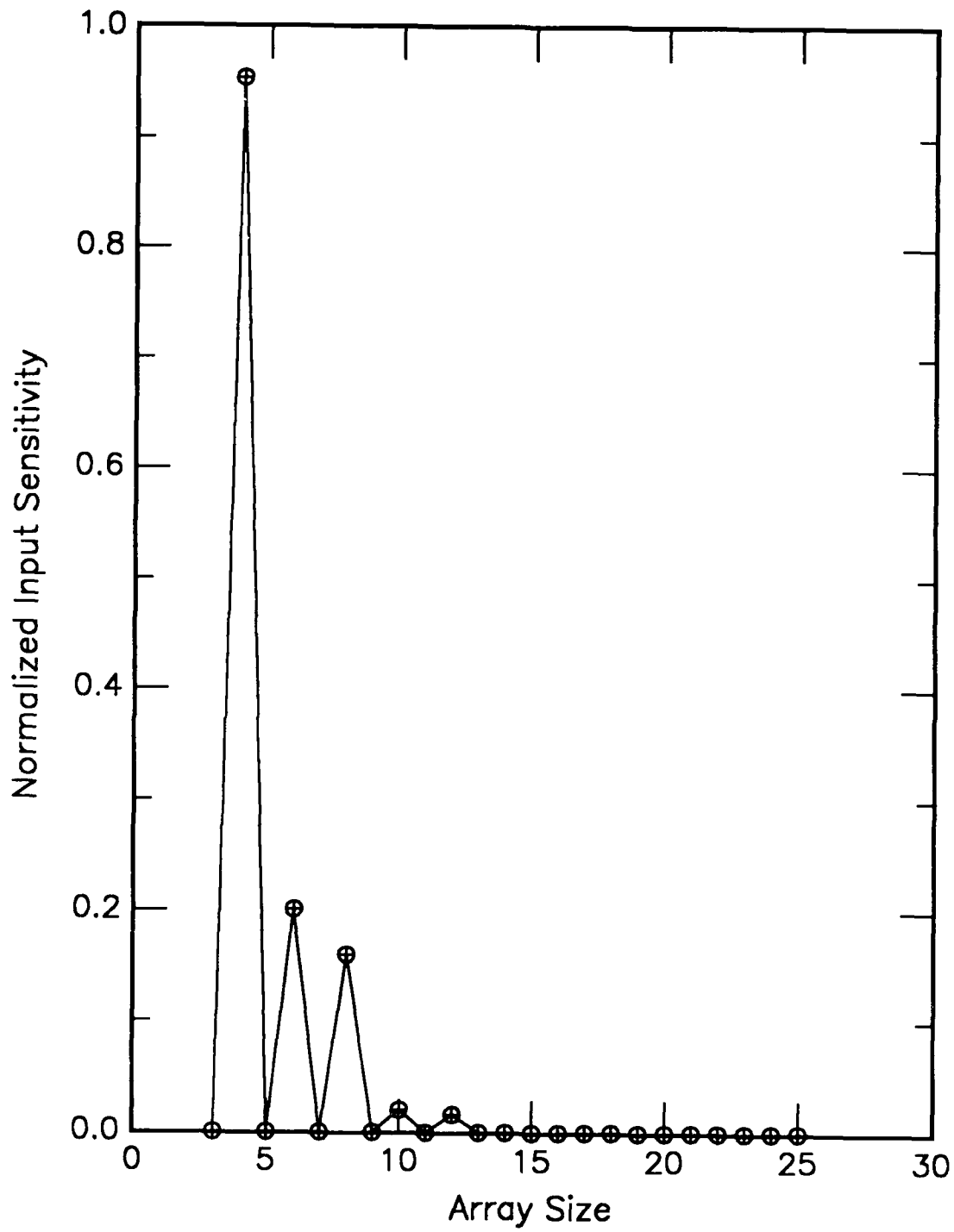


Figure E45. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 105

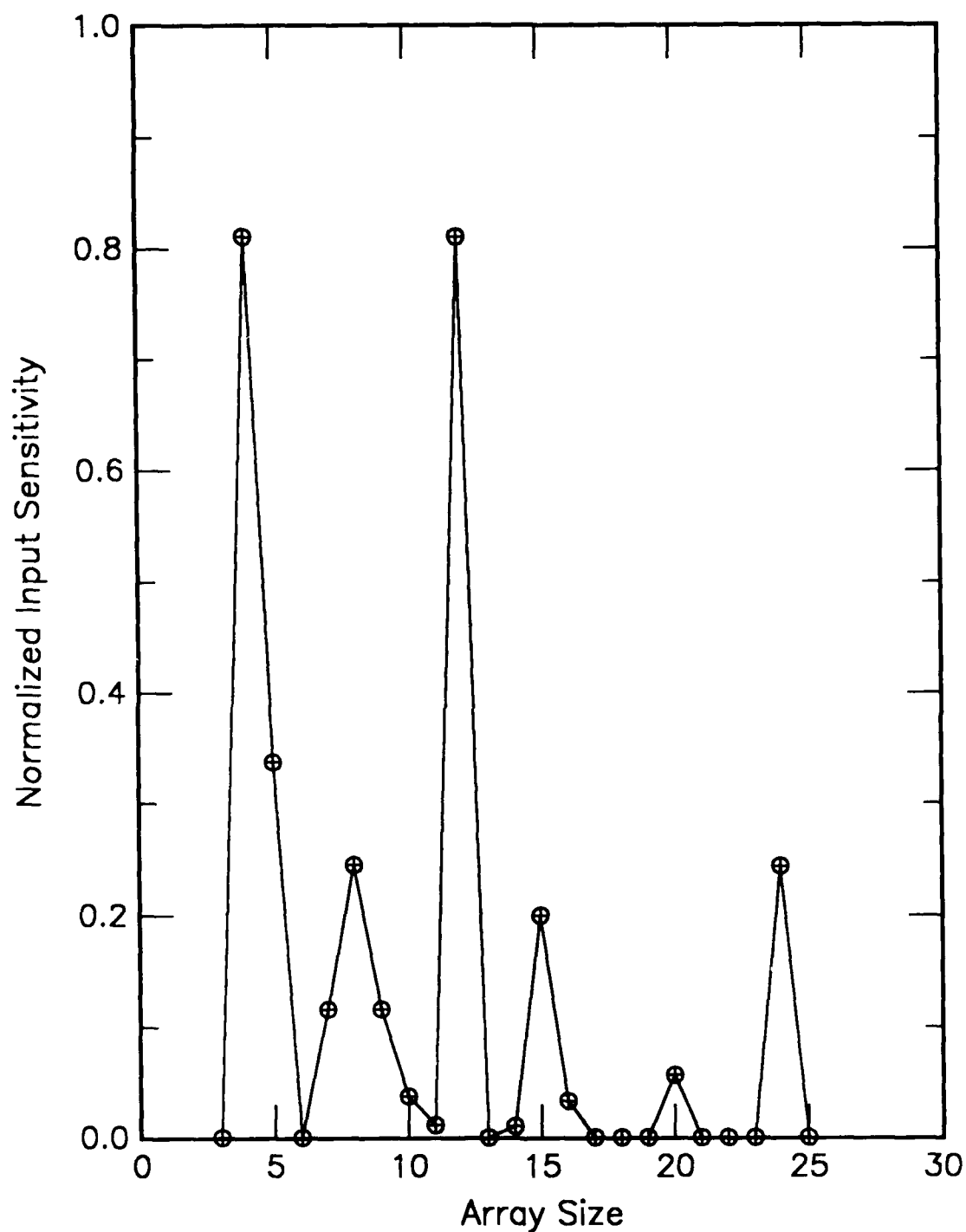


Figure E46. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 106

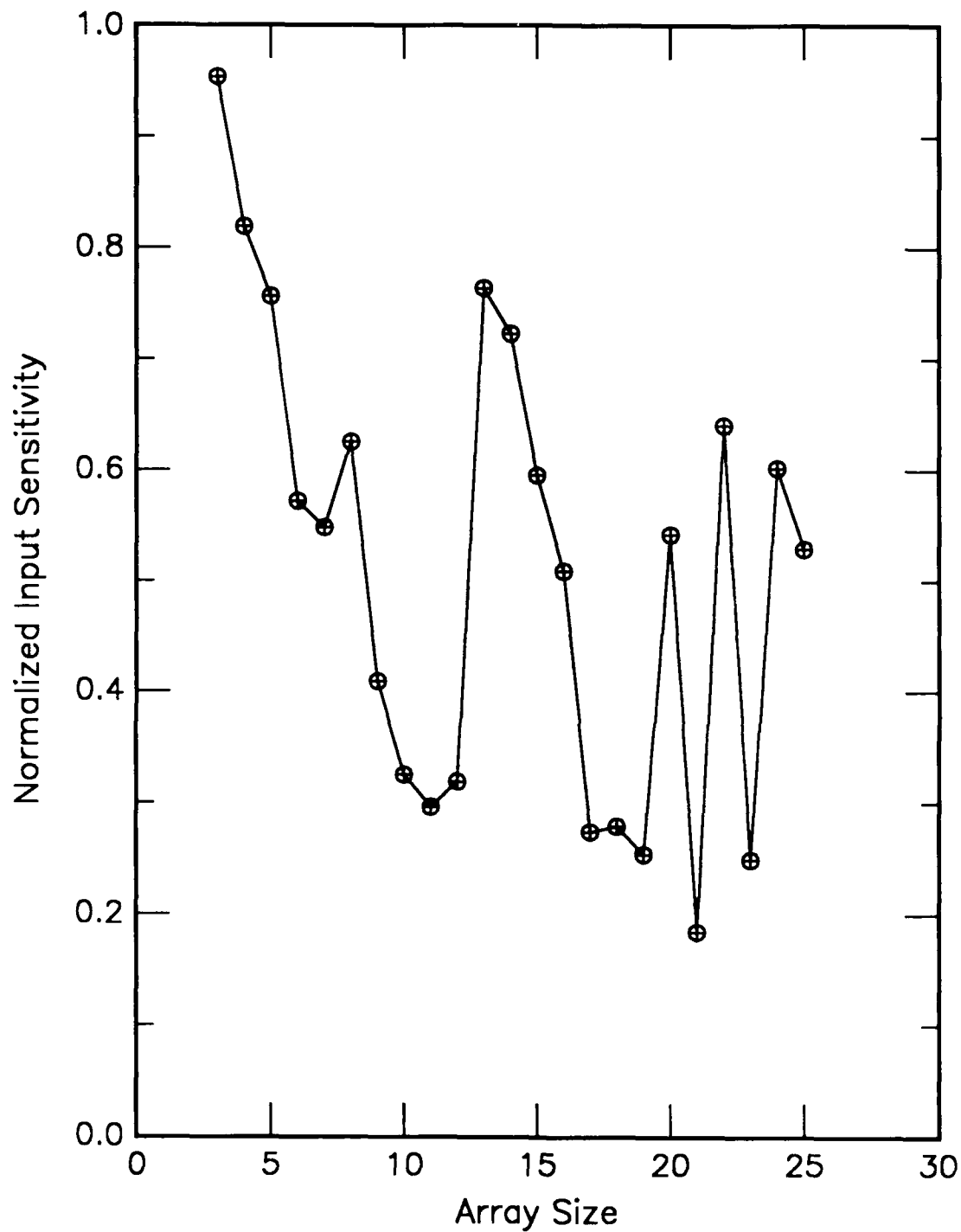


Figure E47. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

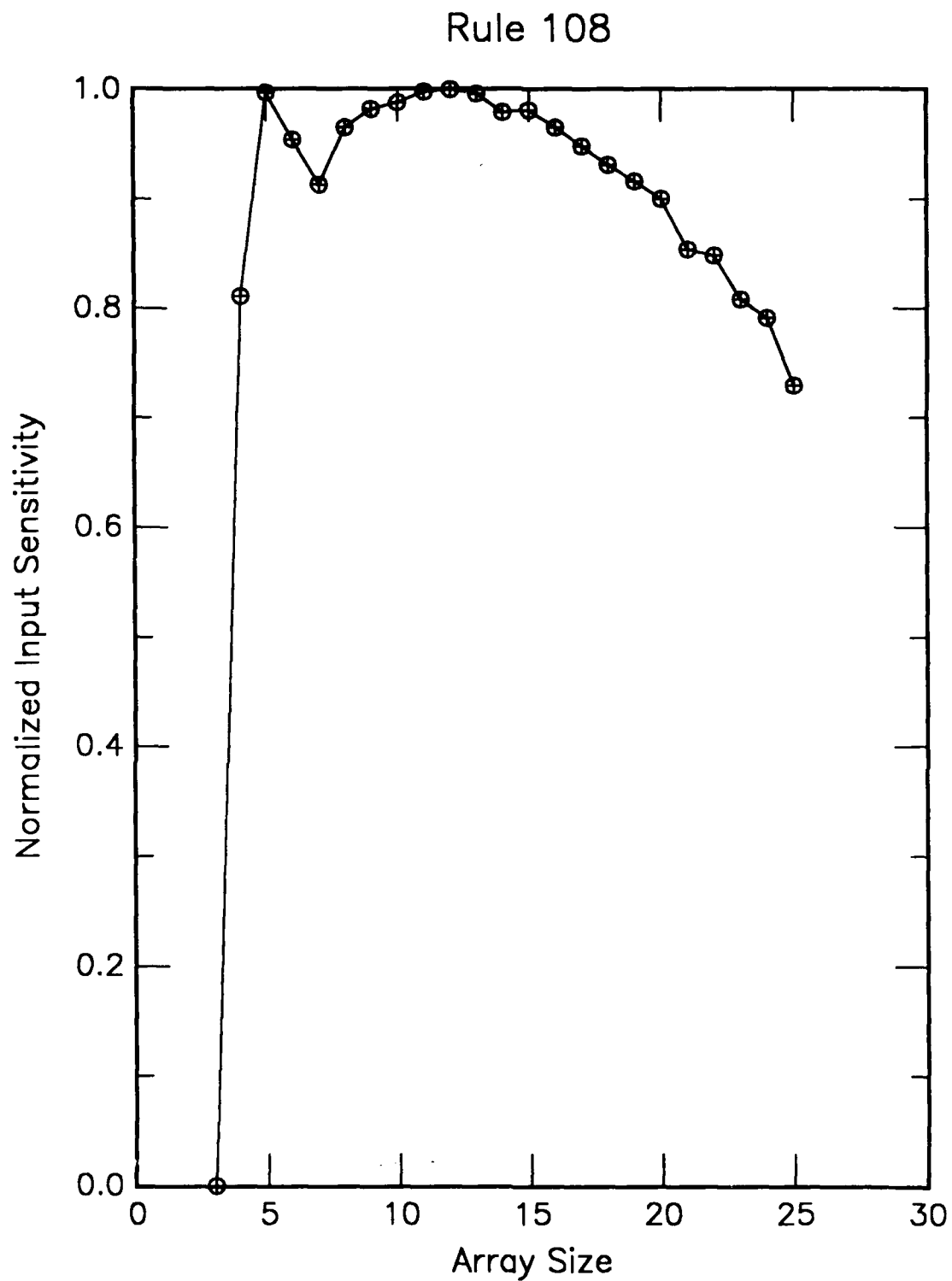


Figure E48. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 110

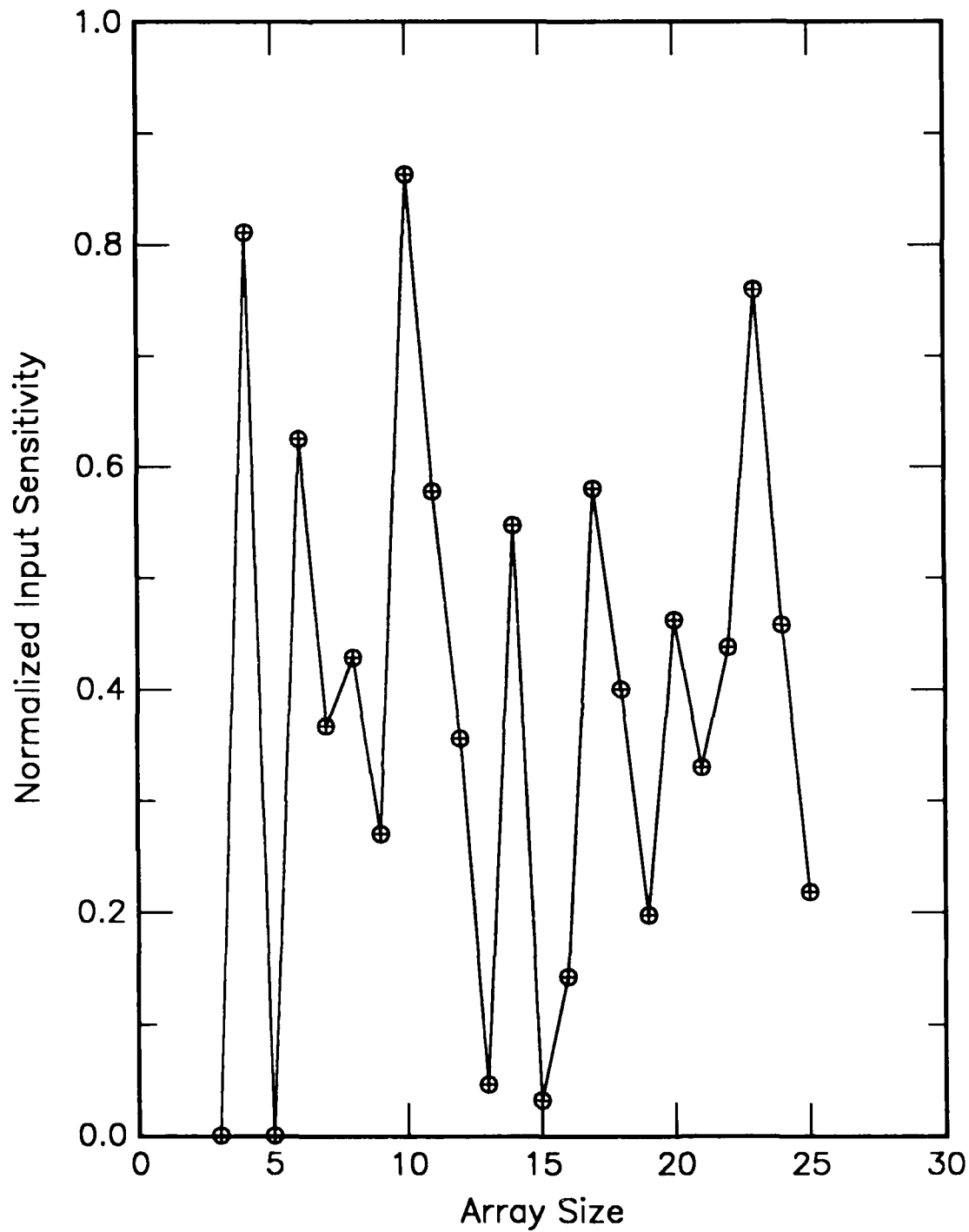


Figure E49. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 122

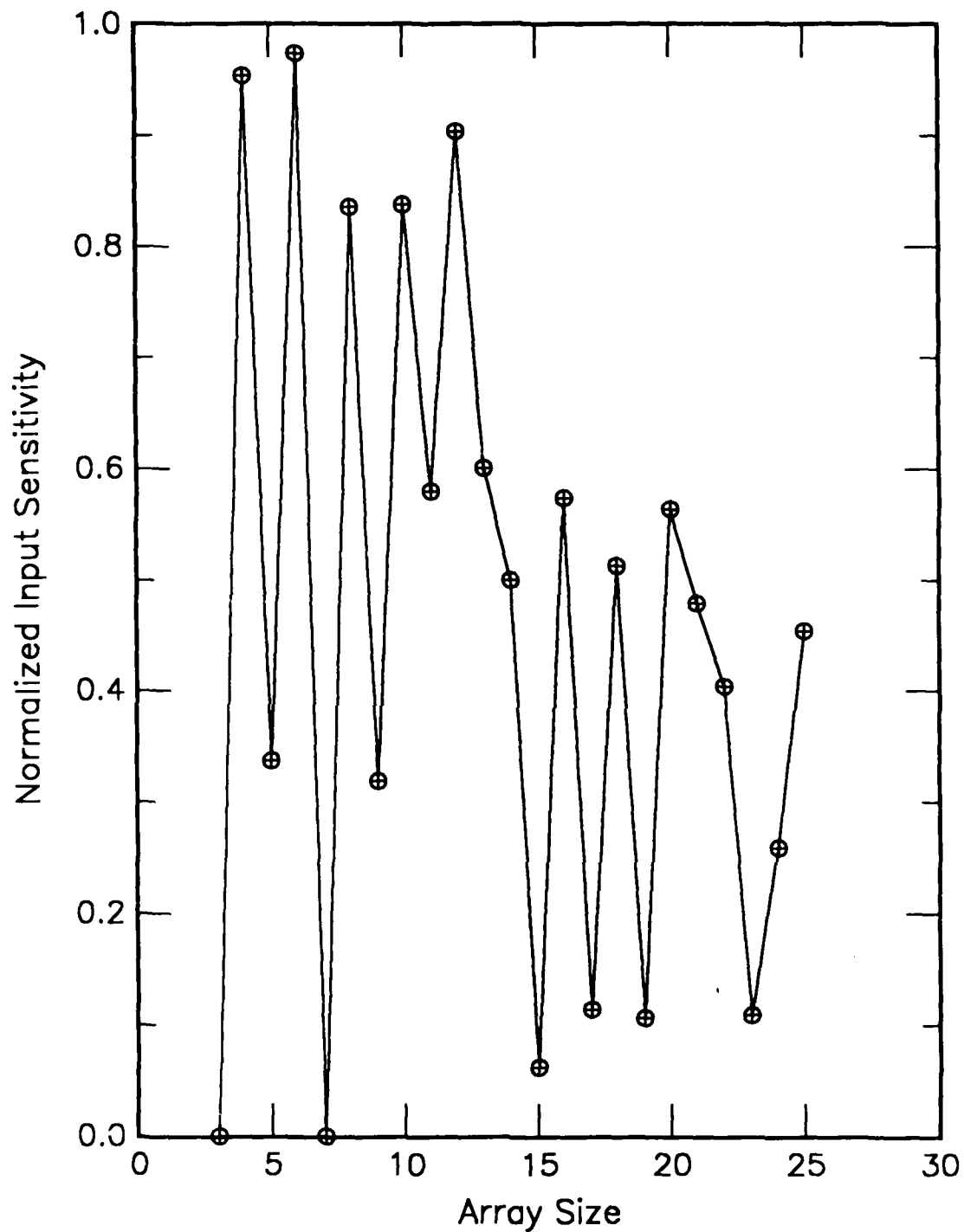


Figure E50. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

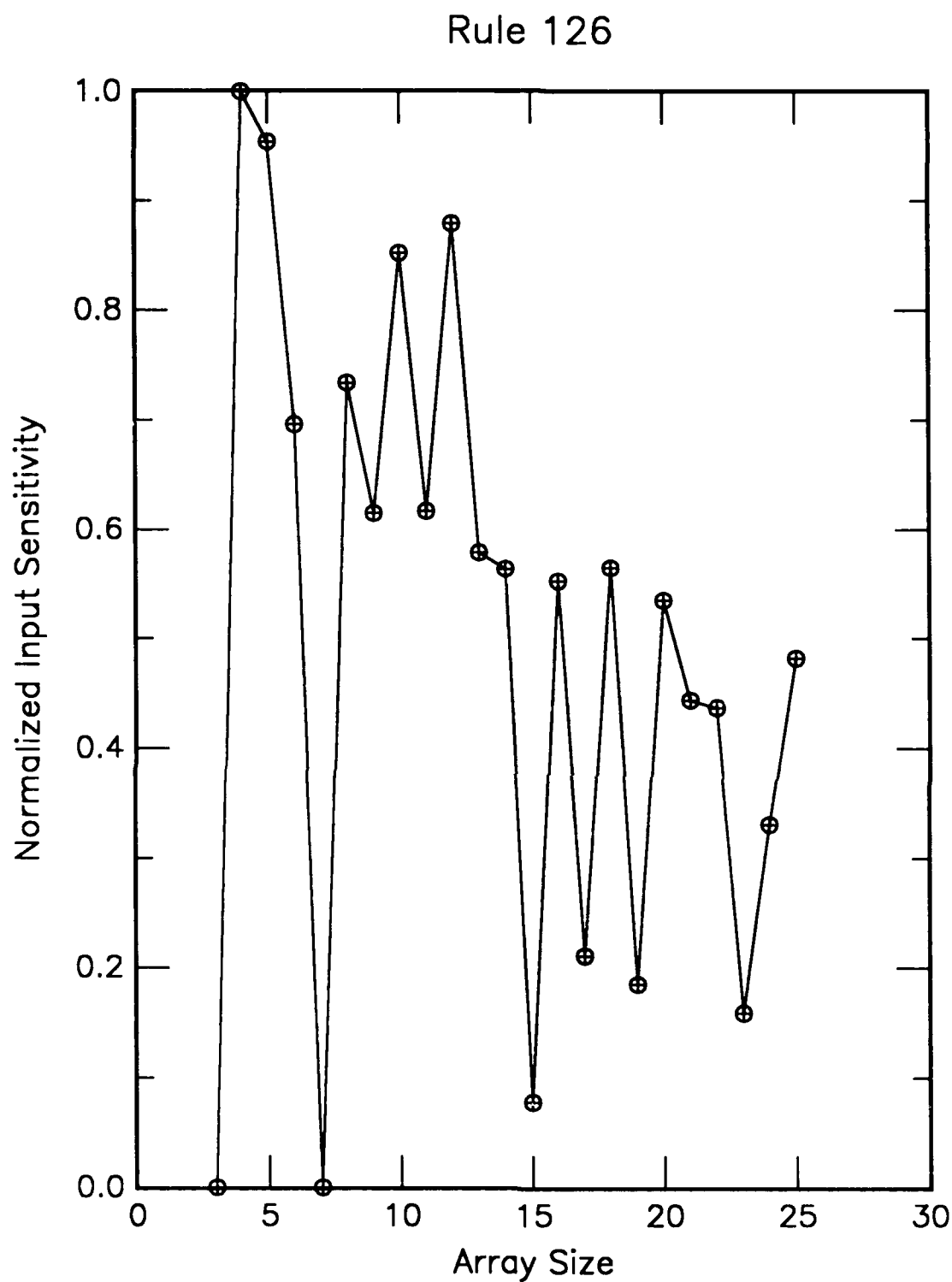


Figure E51. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 130

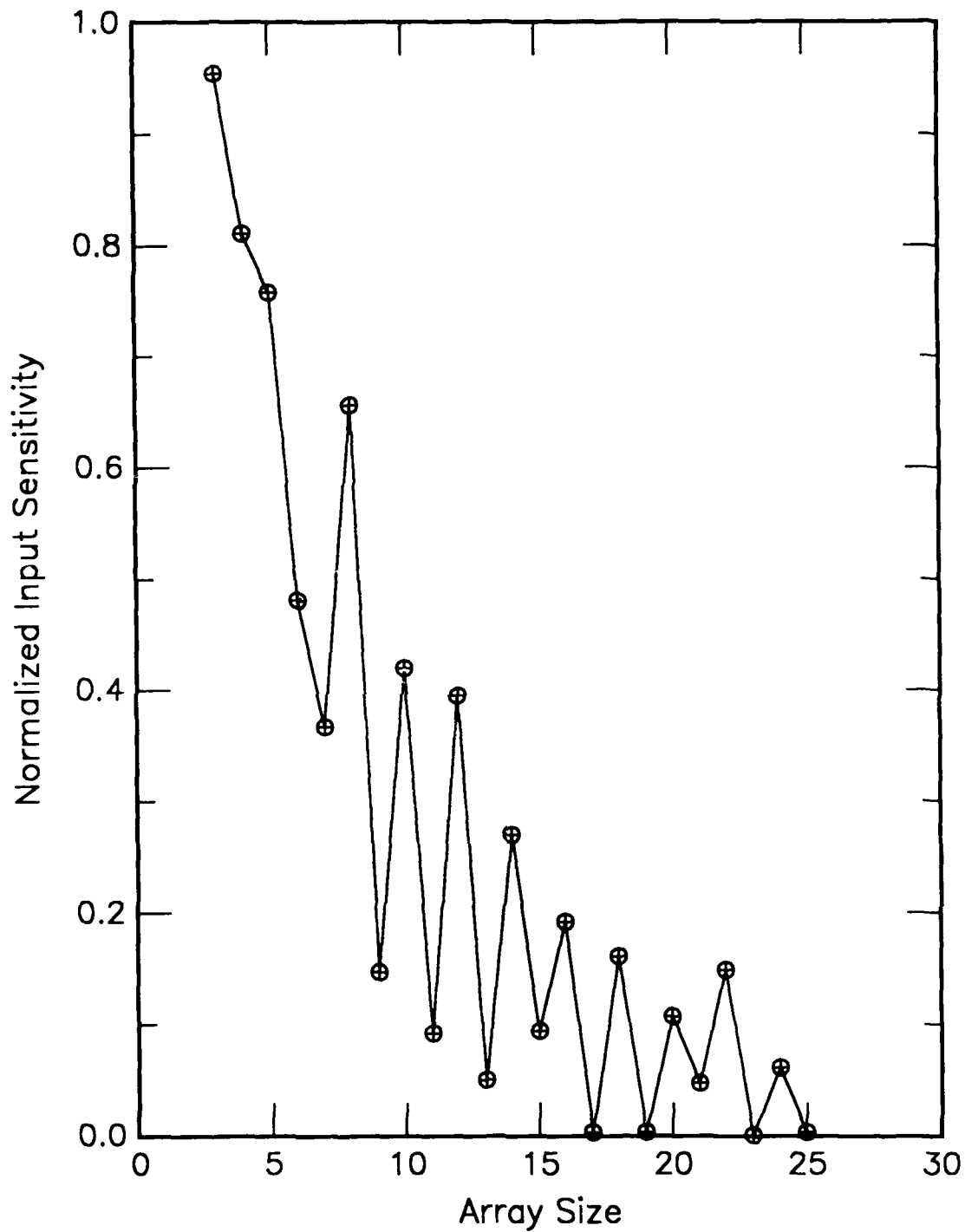


Figure E52. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 134

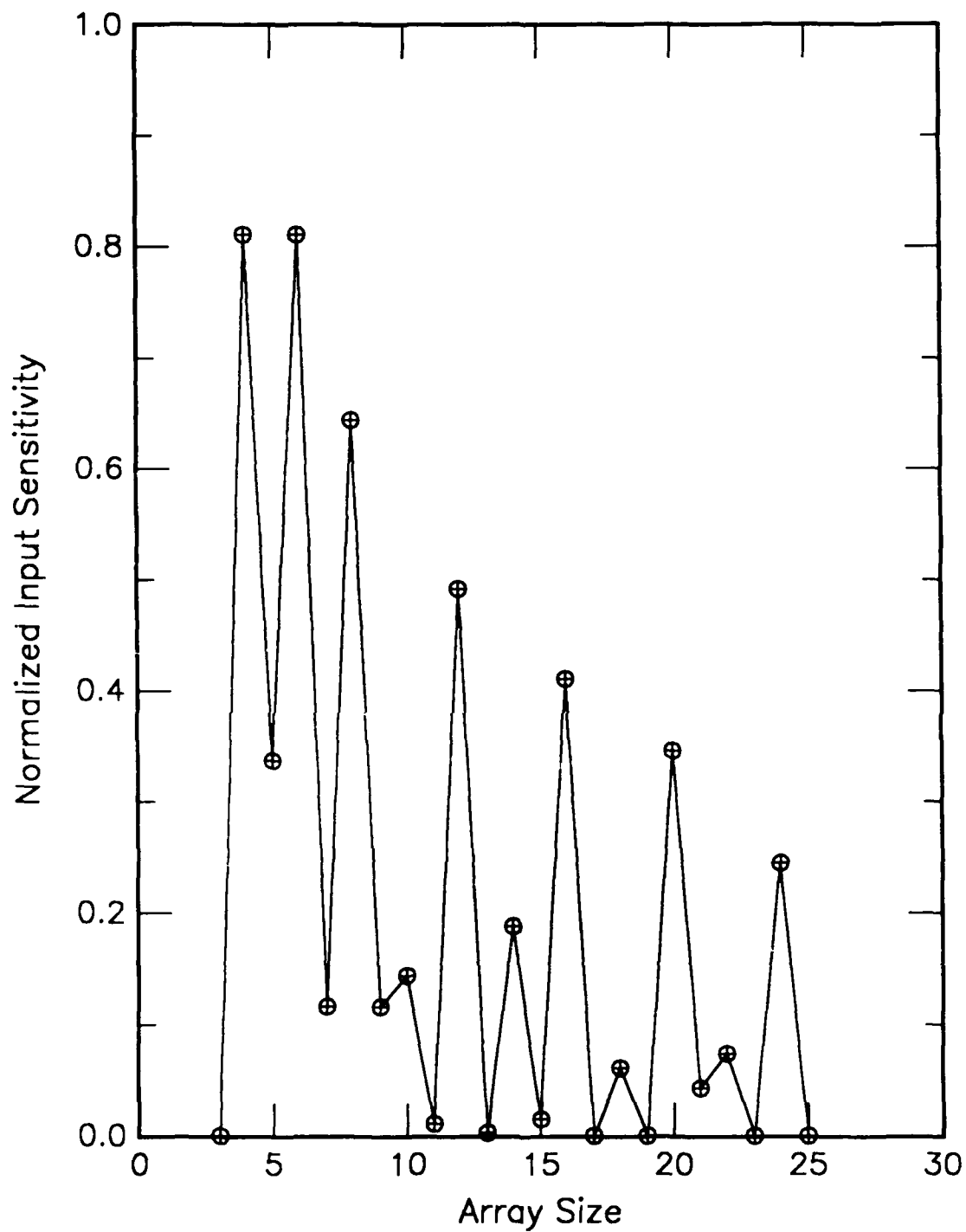


Figure E53. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 138

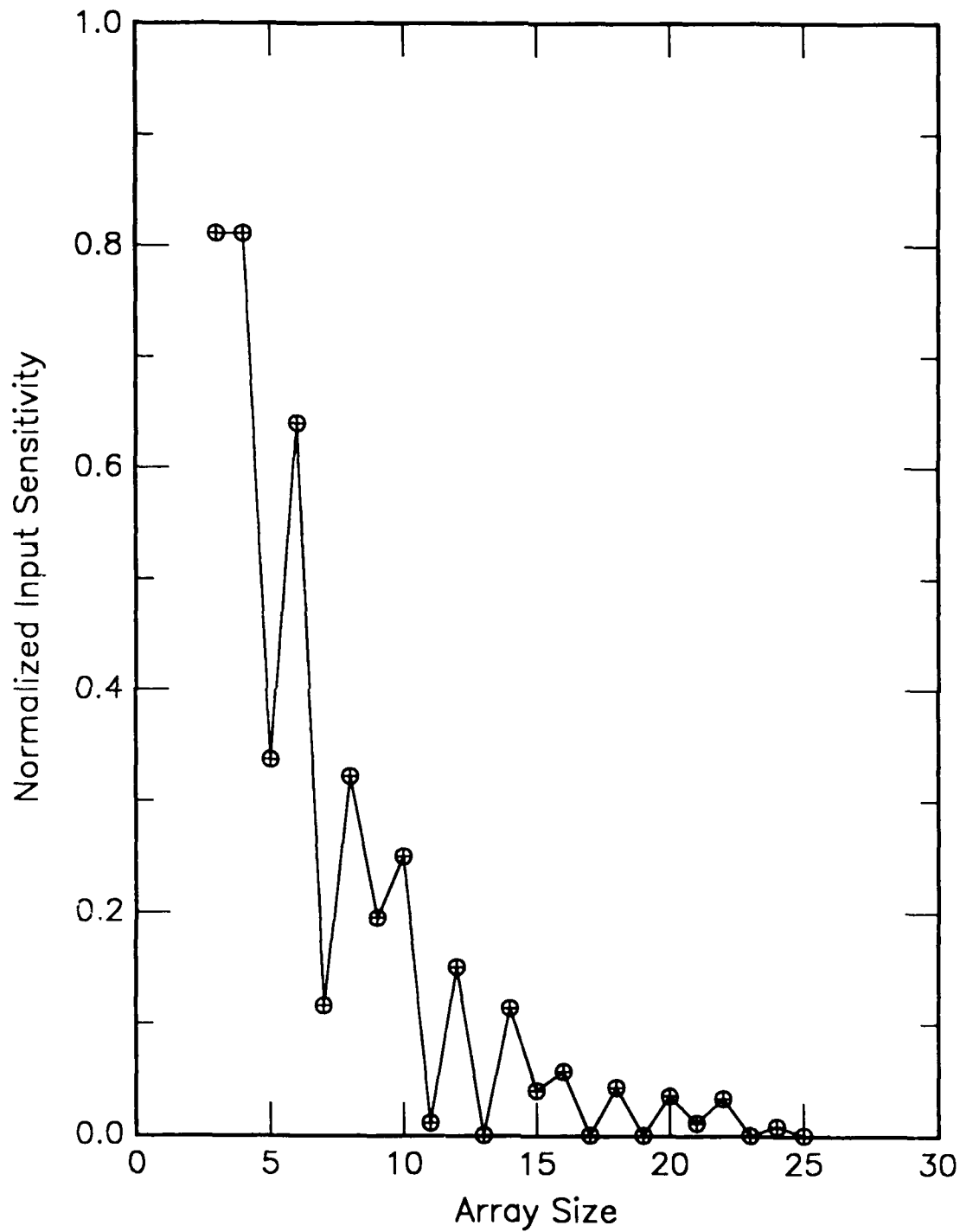


Figure E54. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 146

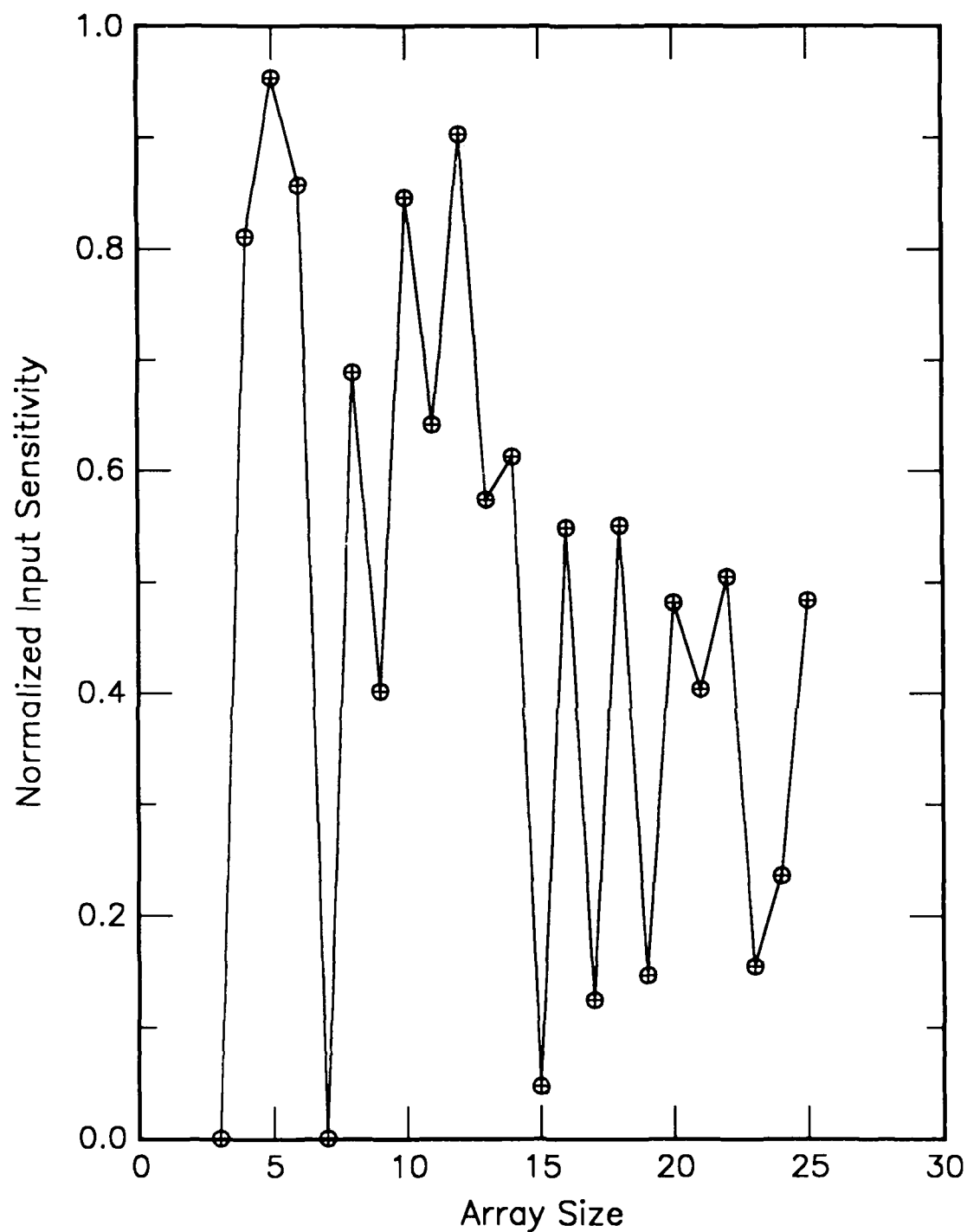


Figure E55. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

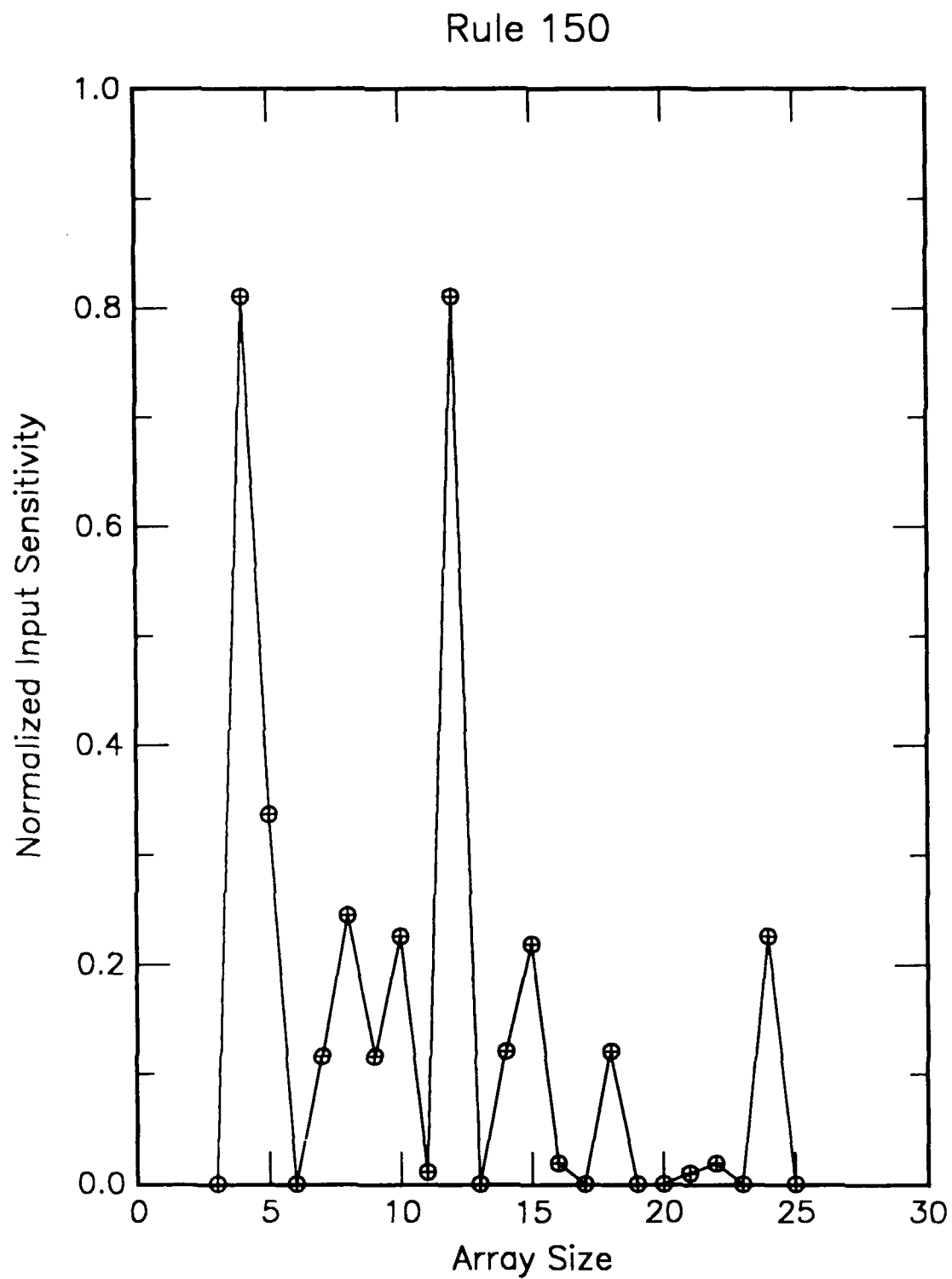


Figure E56. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 152

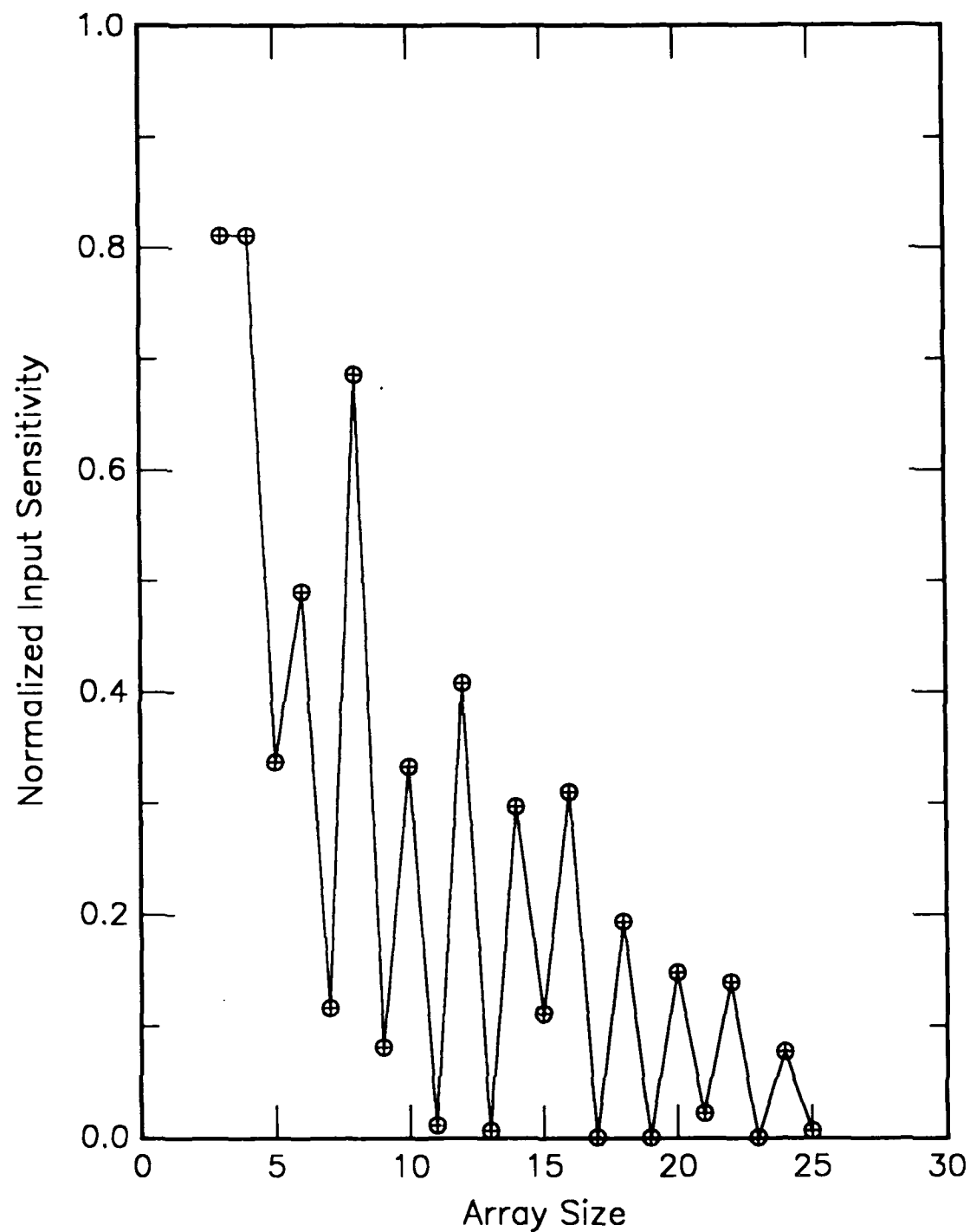


Figure E57. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 154

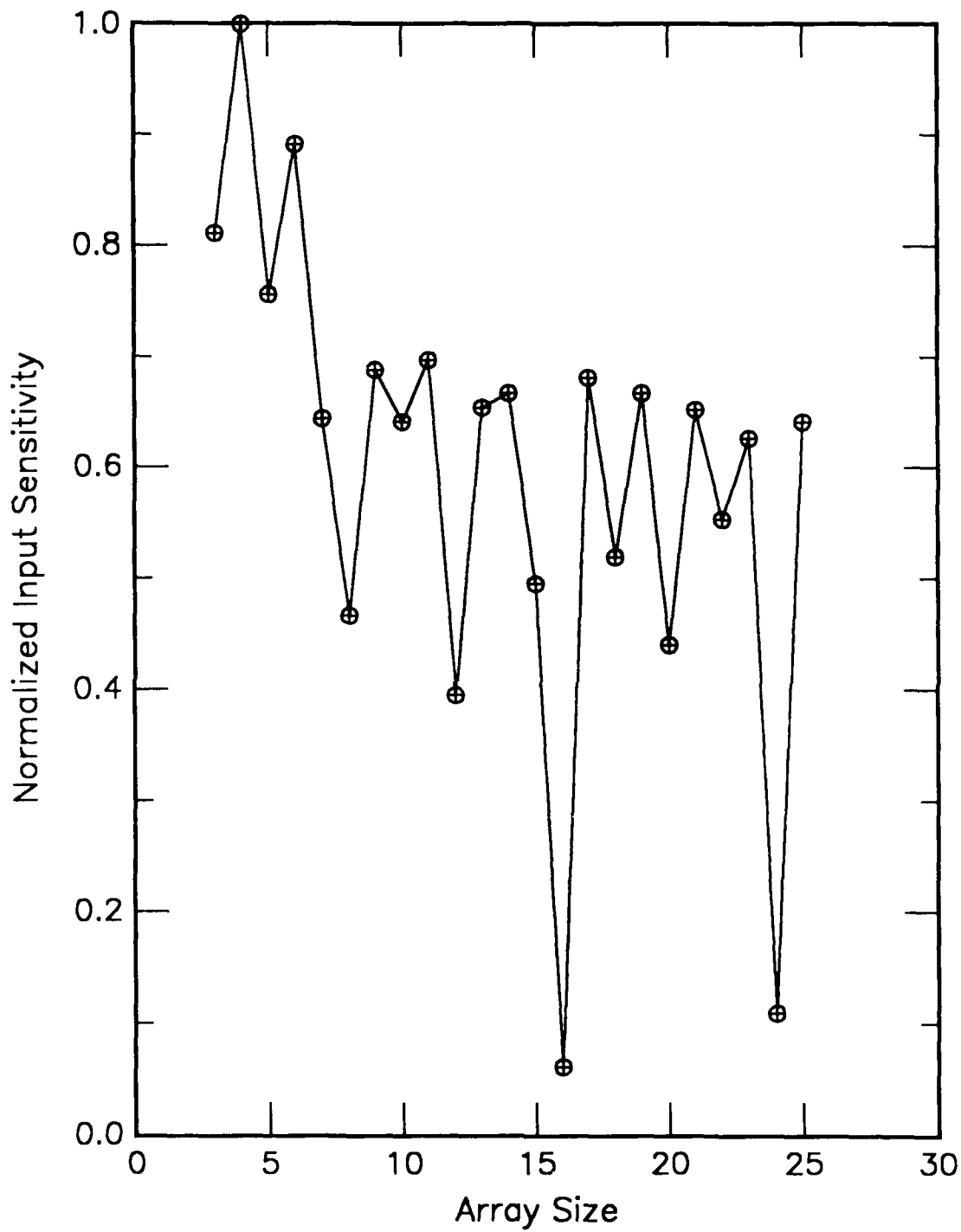


Figure E58. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

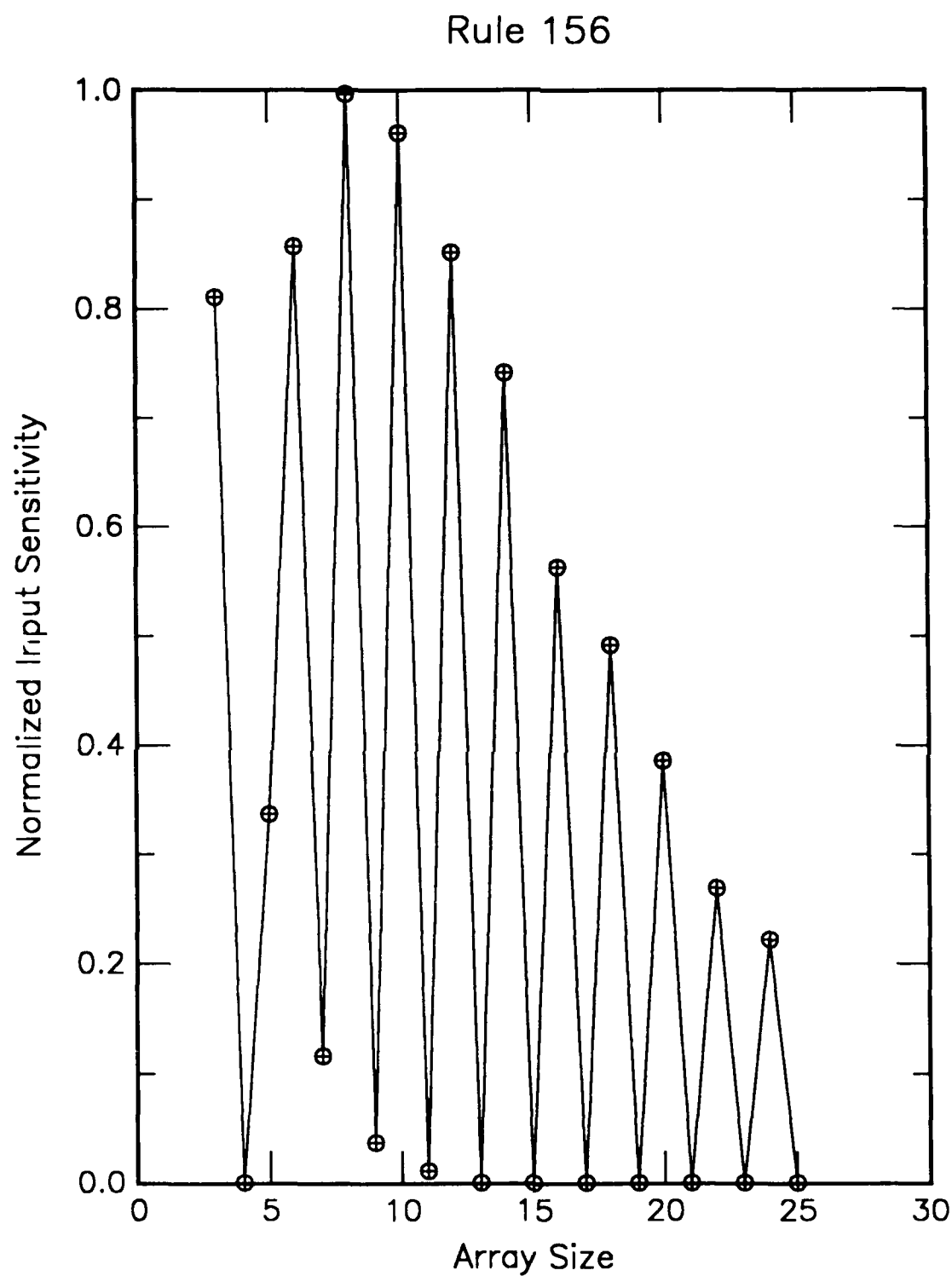


Figure E59. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 160

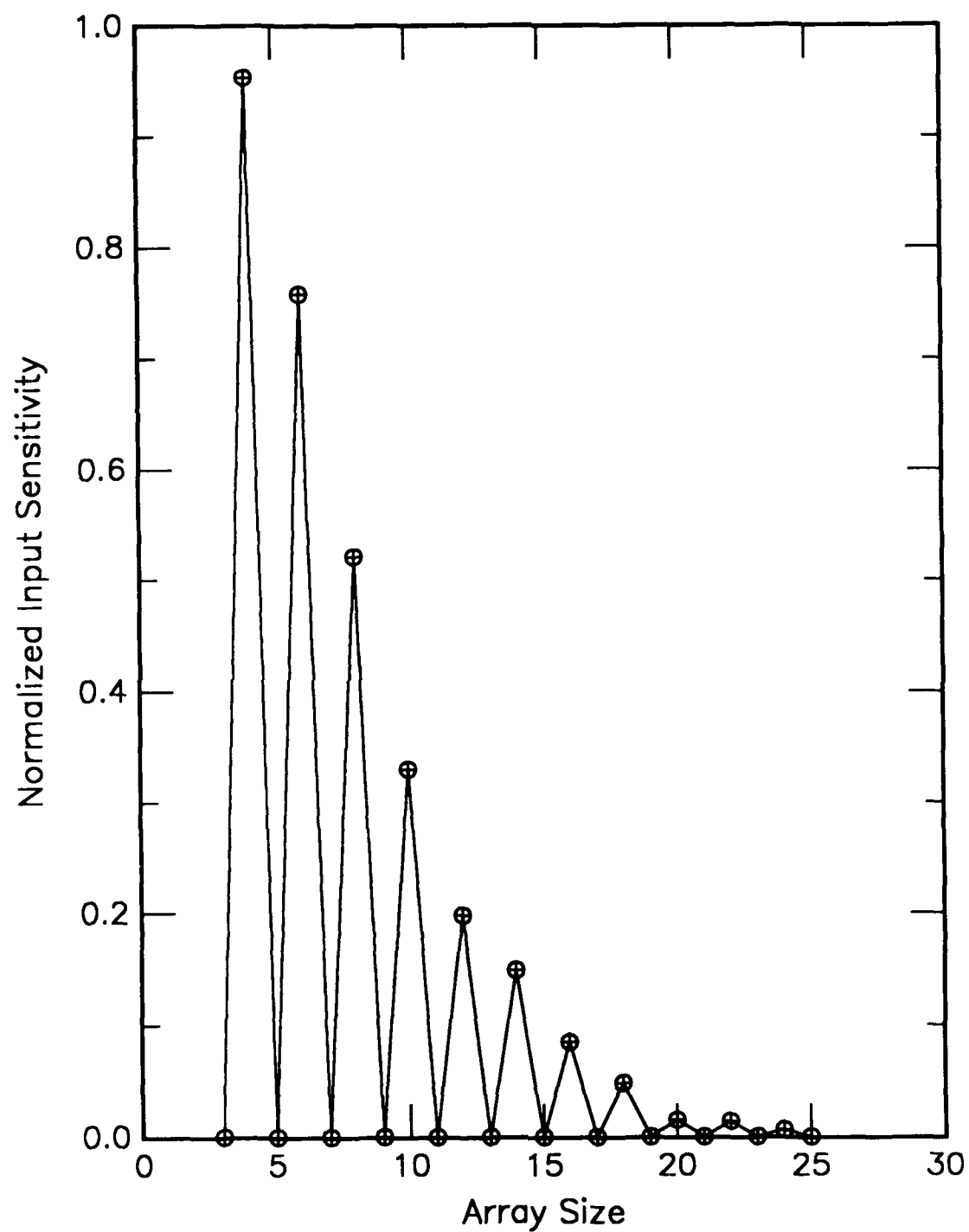


Figure E60. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 162

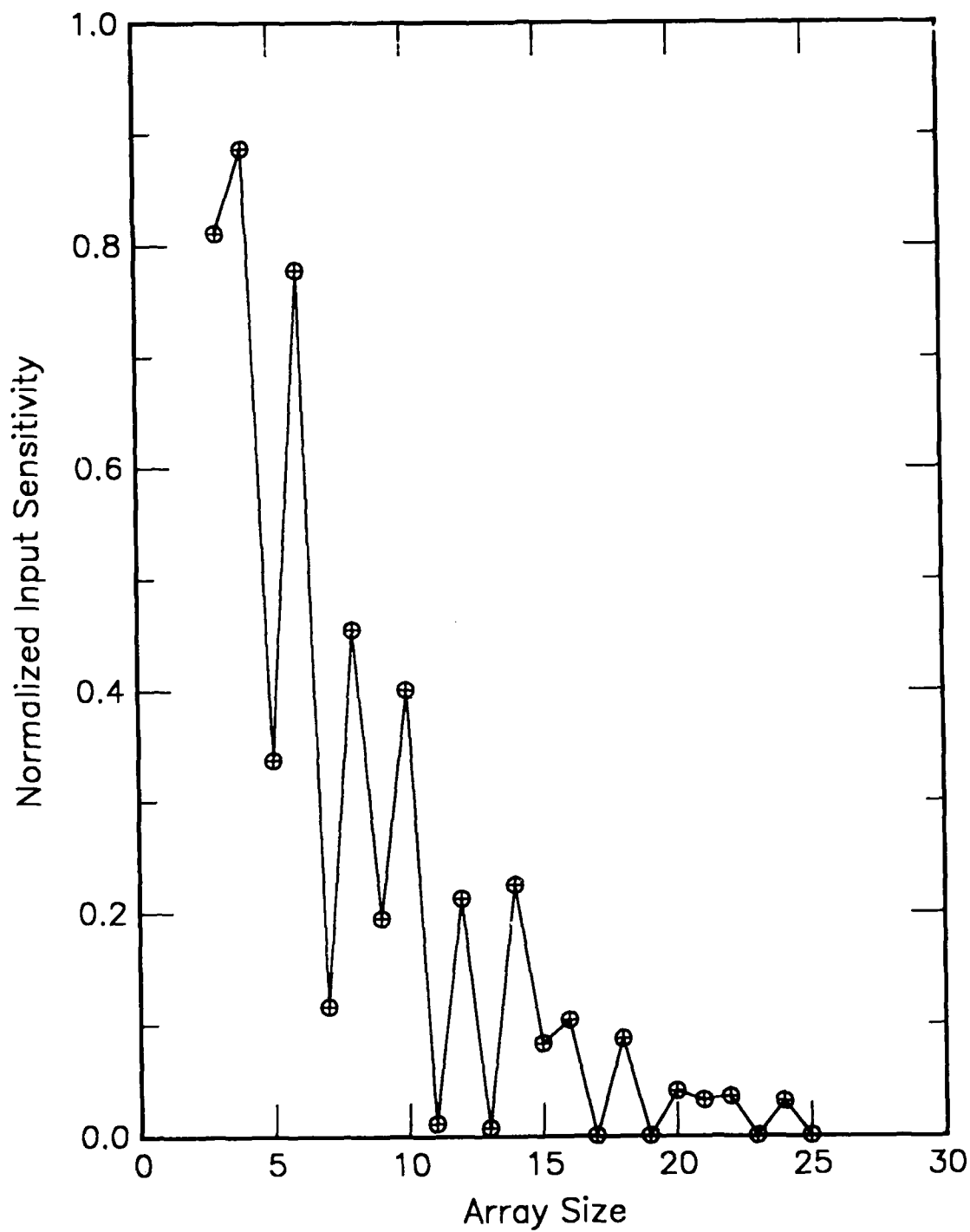


Figure E61. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 164

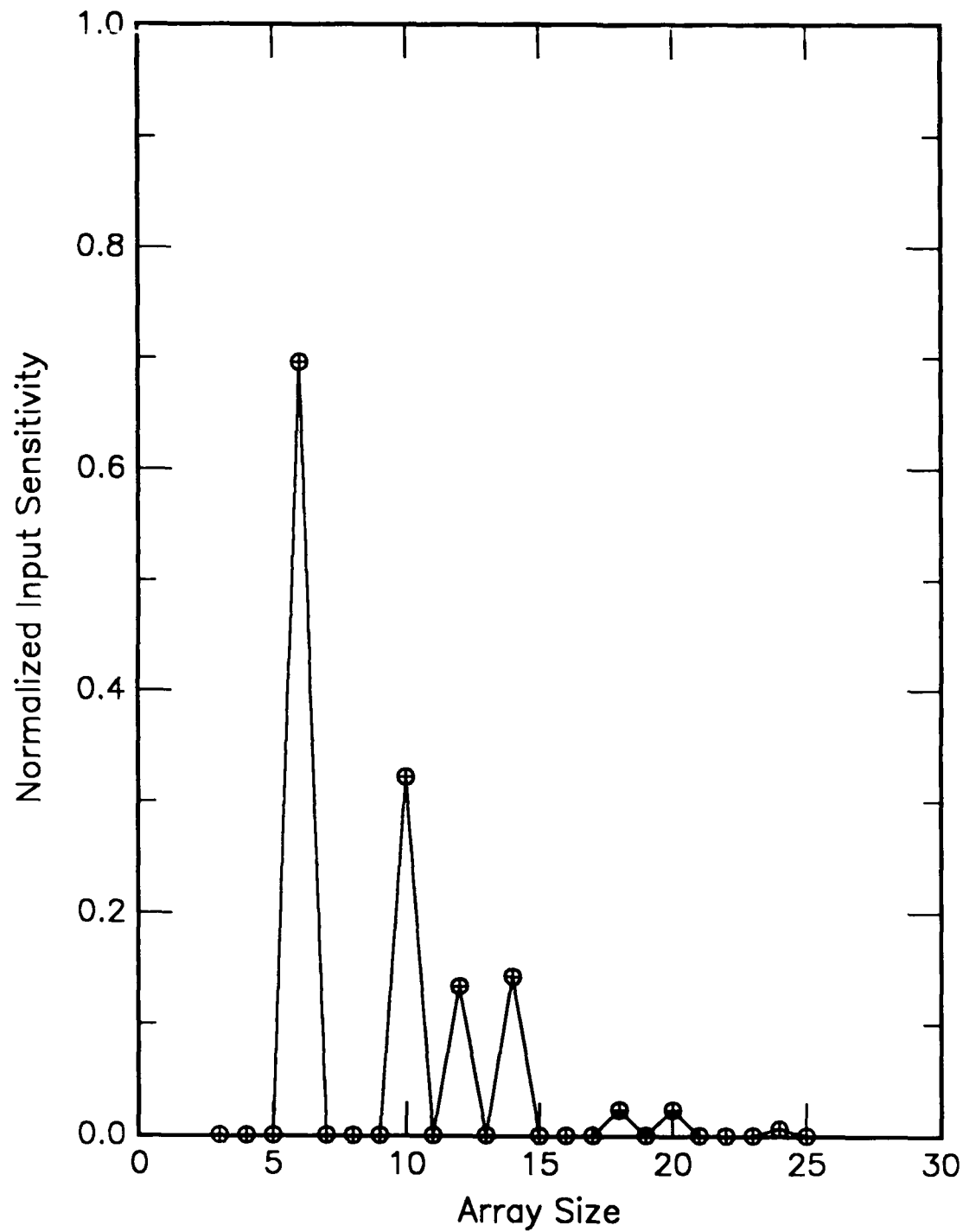


Figure E62. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 168

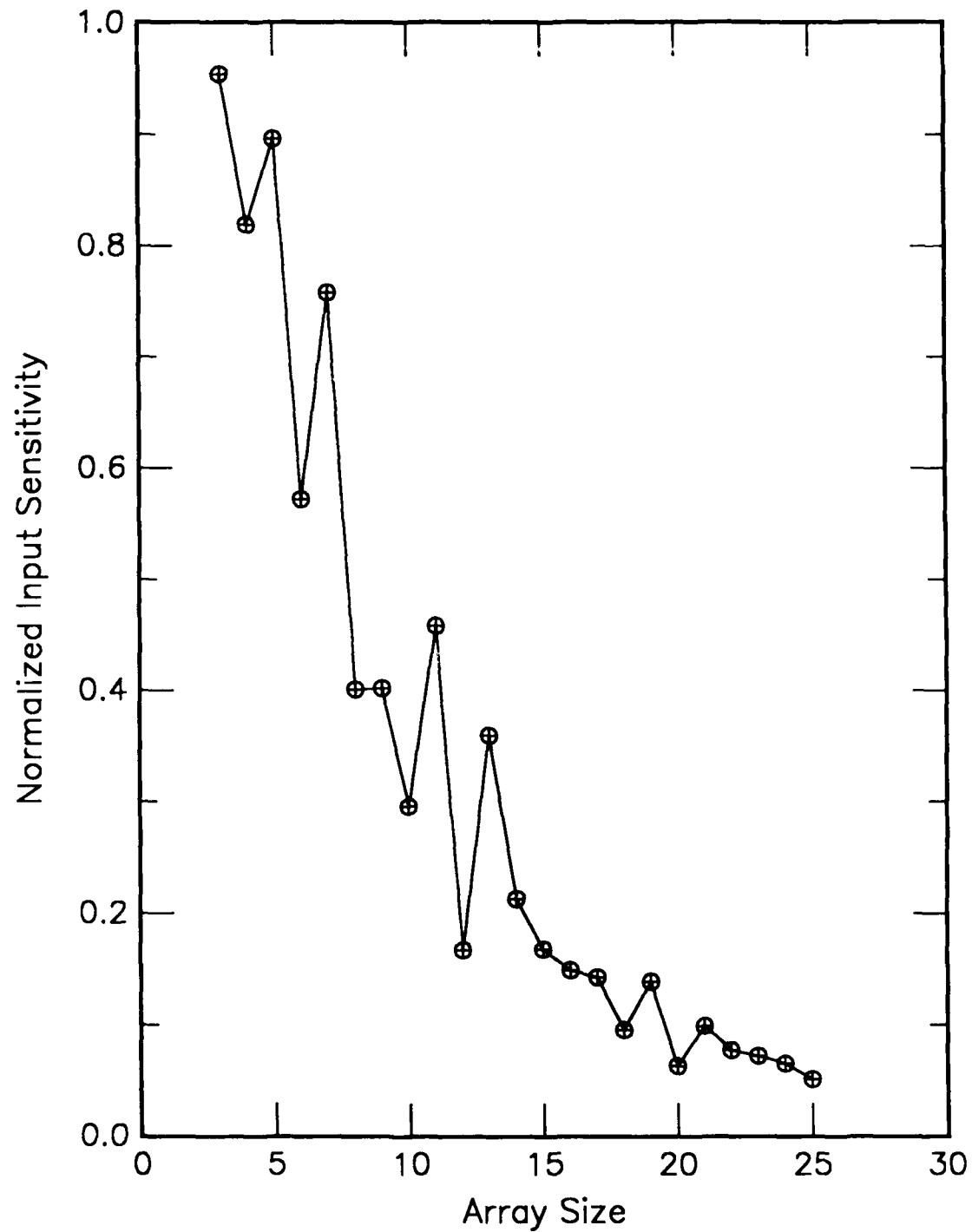


Figure E63. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 170

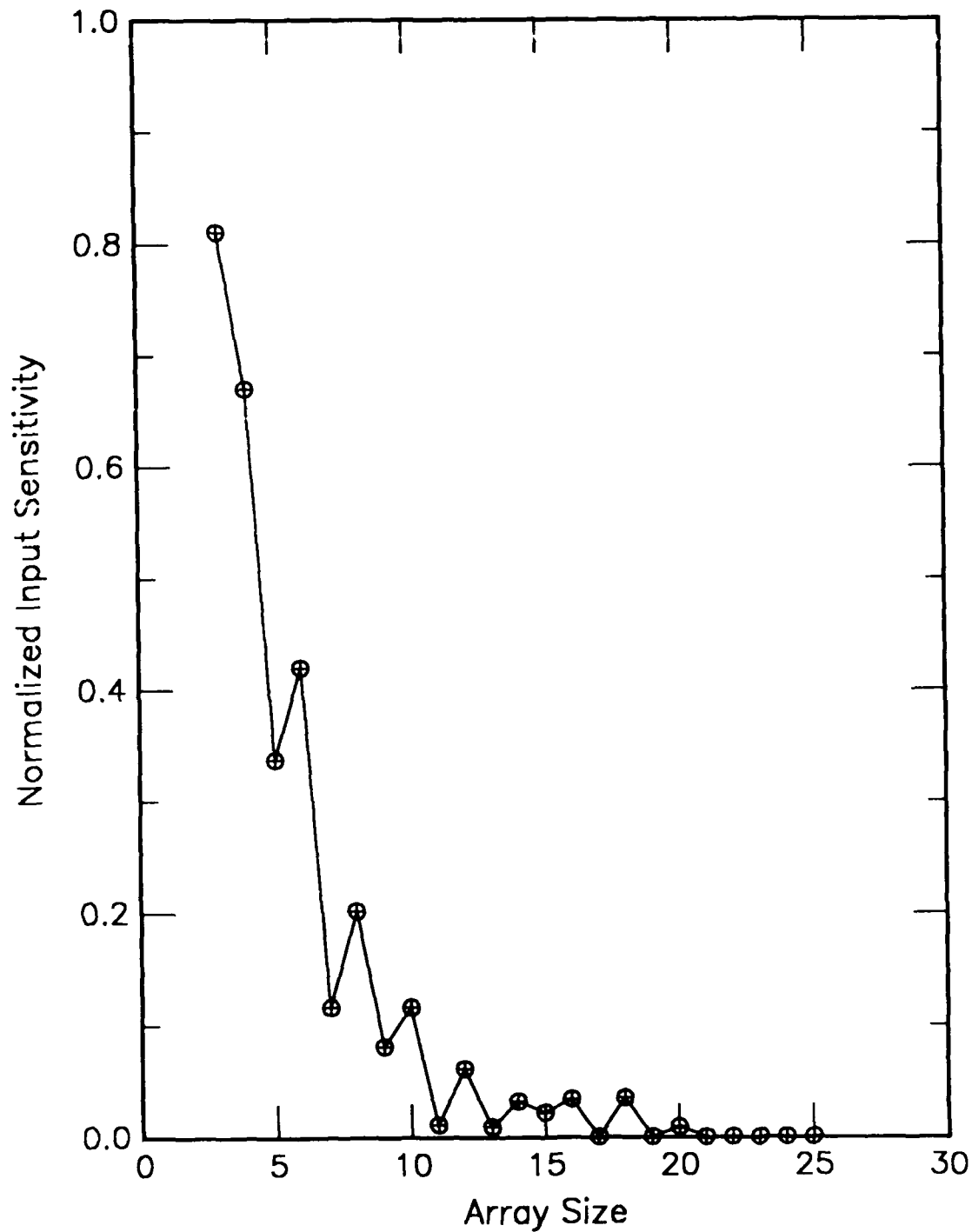


Figure E64. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 172

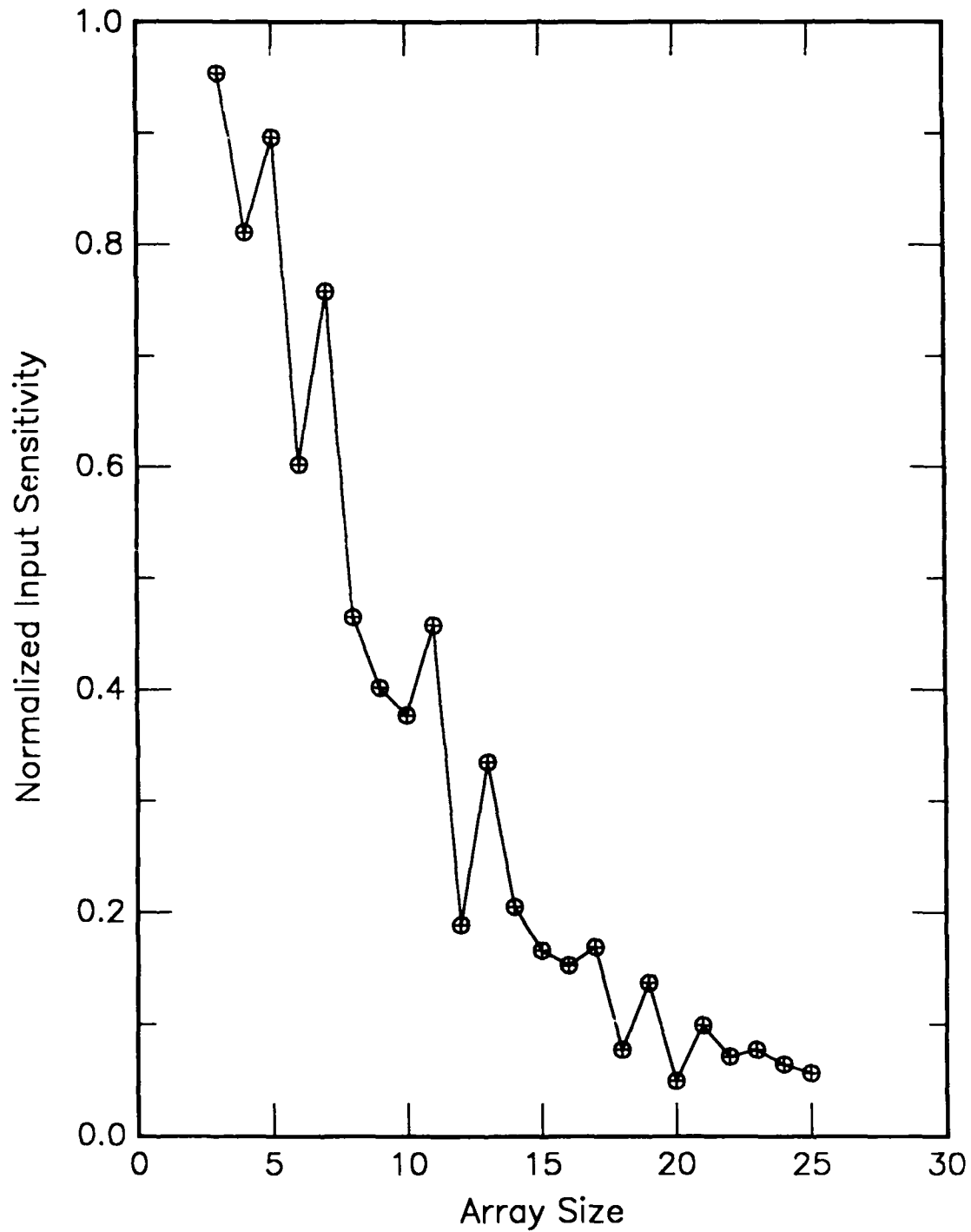


Figure E65. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 178

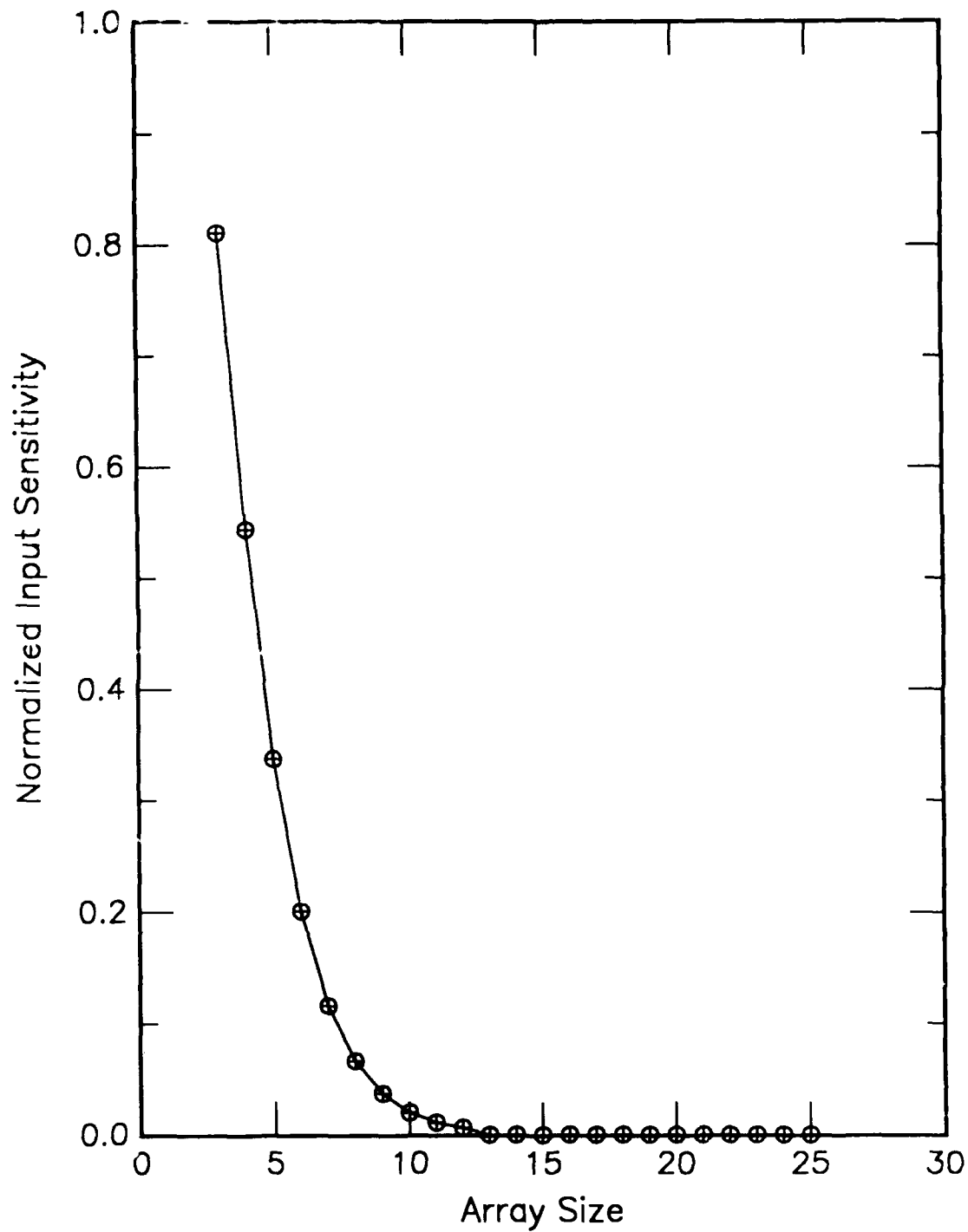


Figure E66. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 184

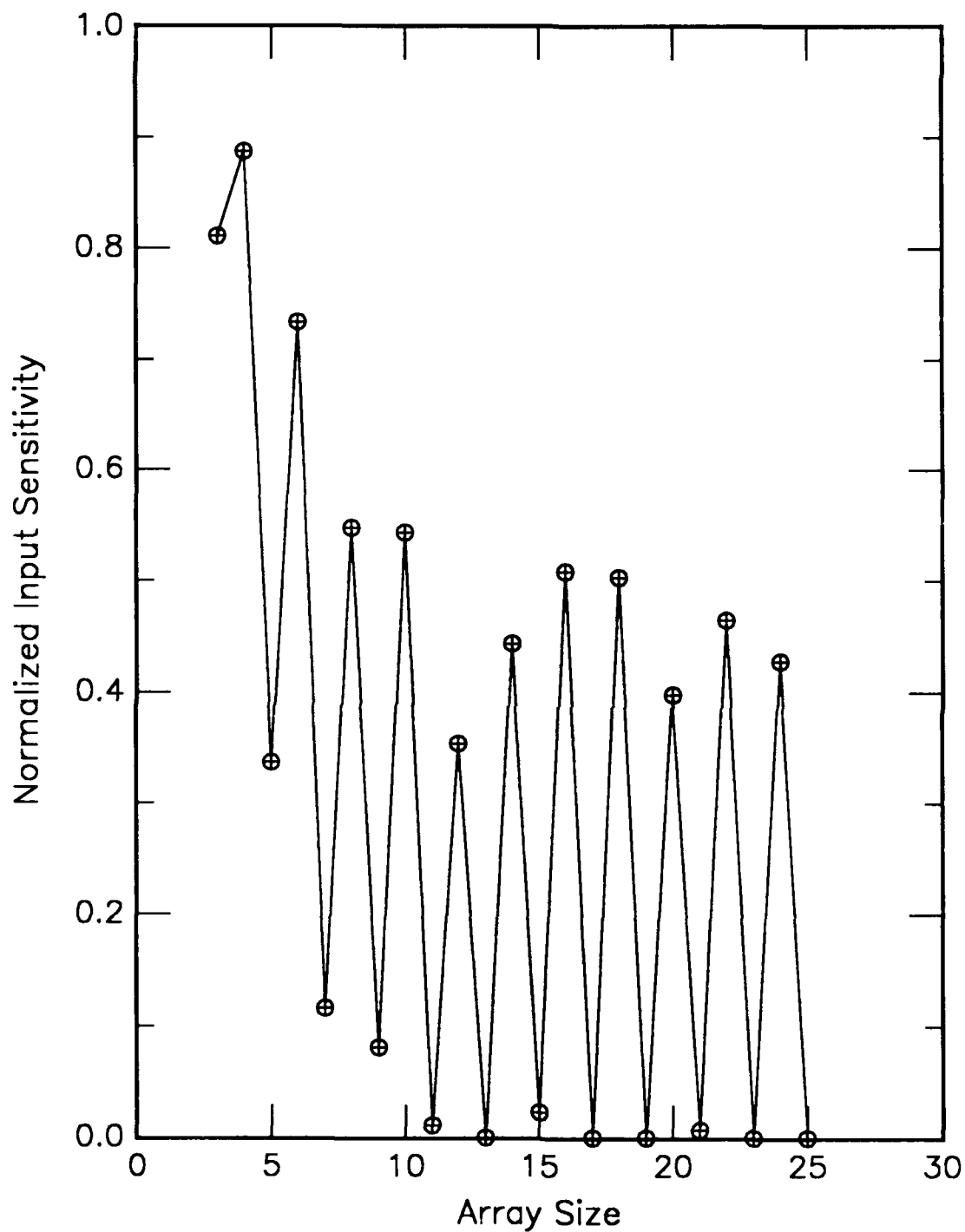


Figure E67. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

Rule 232

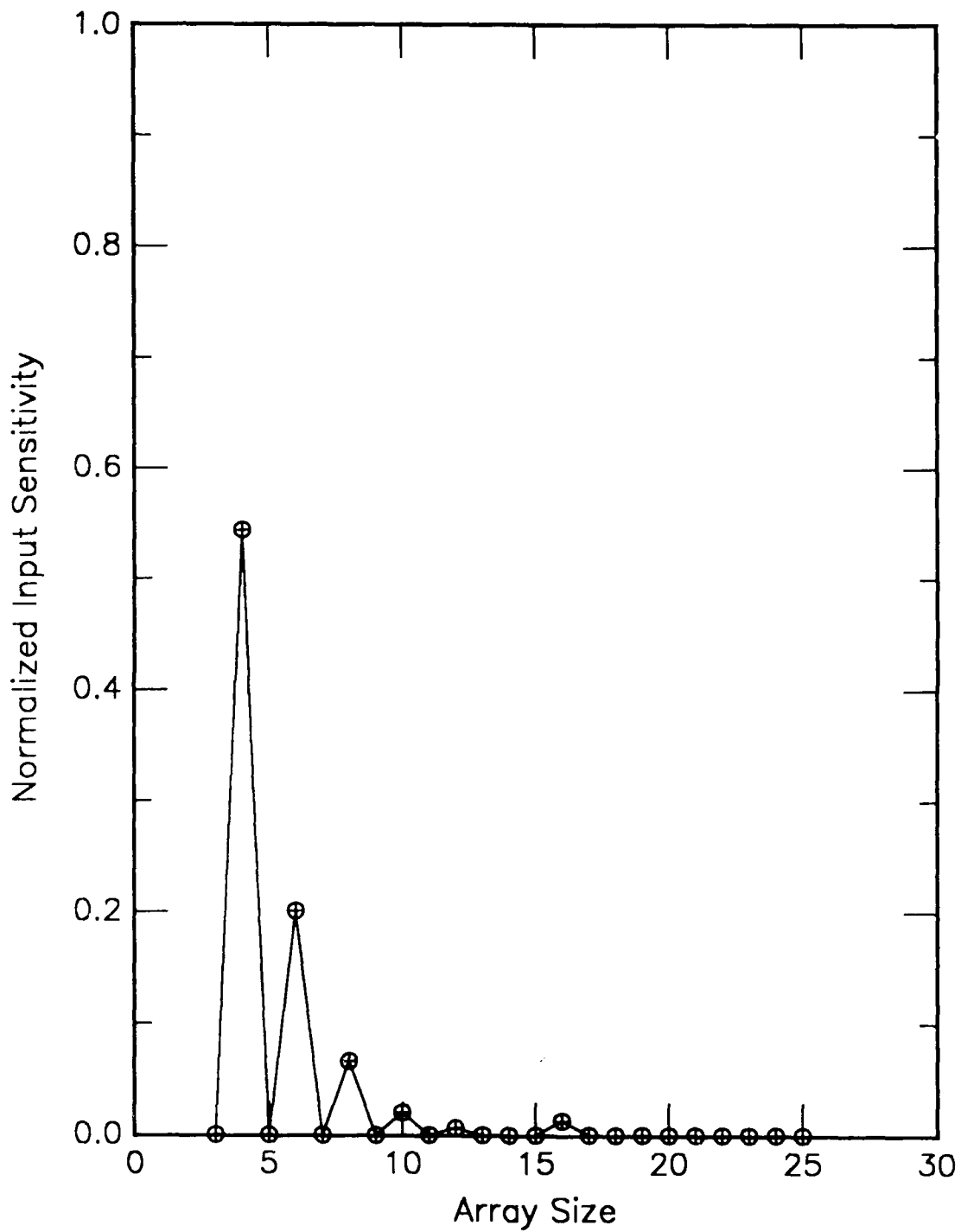


Figure E68. Normalized measure of information required in a cell pattern to predict a future attractor vs array size. Rule is defined in Appendix A.

APPENDIX F

MARKOV MATRIX ELEMENTS FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

APPENDIX F

MARKOV MATRIX ELEMENTS FOR ONE-DIMENSIONAL CELLULAR AUTOMATA

The state attractors of a stochastic cellular automaton may be modeled as discrete Markov Chains. This section presents the Markov Transition Matrix elements for one size of nearest-neighbor coupled, one-dimensional cellular automata. The results comprise two sets of matrix elements. First, the transition matrix elements defined by Equation (5) in Section V are shown as three-dimensional scatter plots. Each figure plots the conditional probability (Z) that noise will cause the dynamics of a particular automaton to jump from one attractor (X) to another (Y). Second, the Limiting Transition Matrices were tabulated and are shown in the accompanying tables. The tables are labeled with a code that specifies the dimensionality, size, rule number, and boundary conditions associated with each particular table. For all of these data, the dimensionality is one (1D), the array size is 25 (L25), and the boundary conditions are periodic (BP). Also shown in the tables are the initial attractor probabilities $V_i(0)$ and the long-term attractor probability distributions. These data quantify the fault tolerance of one-dimensional cellular automata to low rates of event upset.

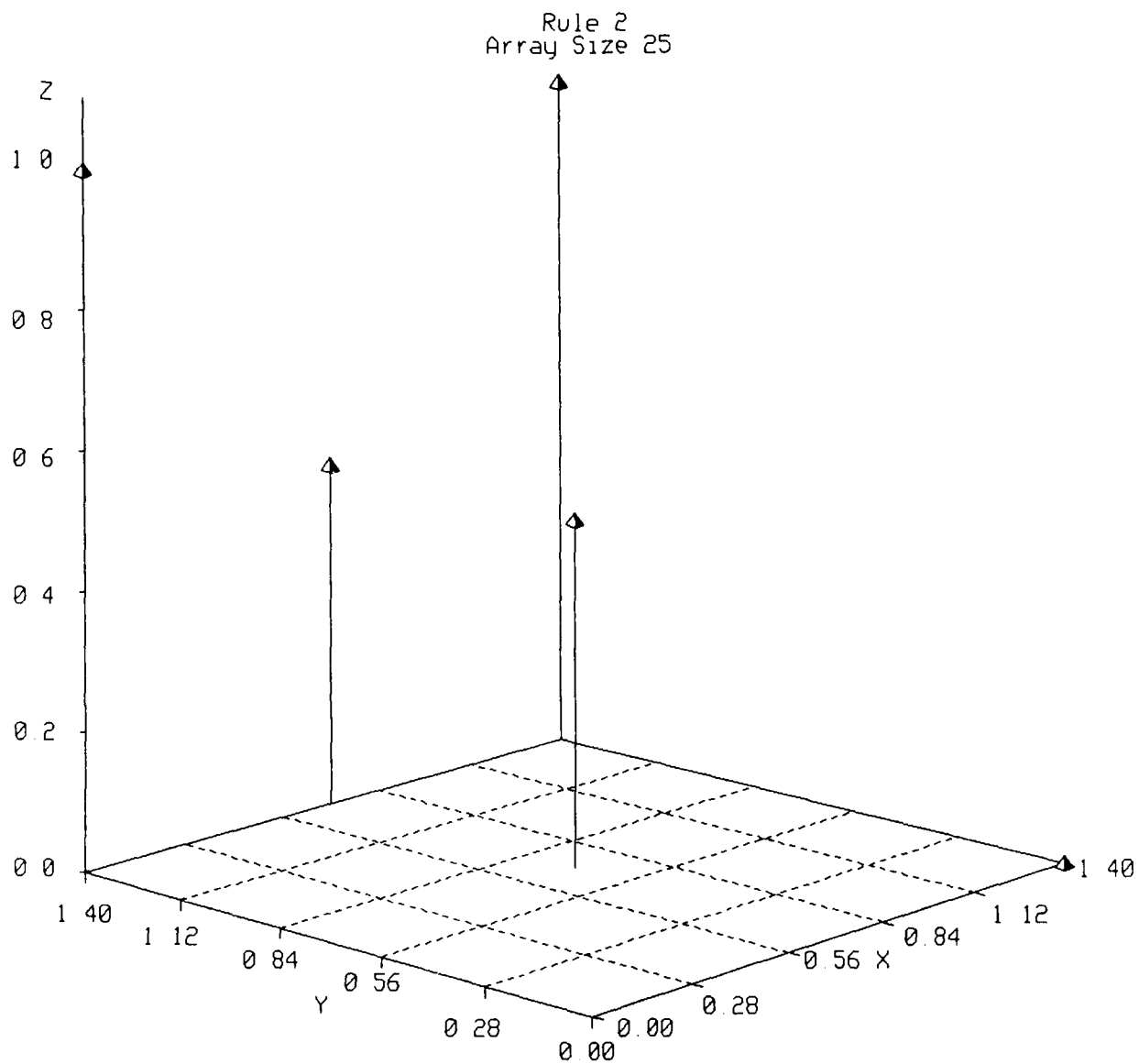
We have presented only the results for automata consisting of a closed loop of 25 cells. Due to the large amount of data collected, it was impossible to show results for all array sizes studied under the program. However, this length of automaton adequately represents all of the simulated behaviors observed. The complete analysis of the Markov properties of variable sized automata is being prepared for publication at this time.

Of the 256 possible nearest-neighbor cell rules, only 88 rule operations are independent. In addition, several of the independent rules lead to constant sensitivity factors. In particular, we did not include any results for rules that were characterized by a single matrix element.

The fault tolerance of a given rule may be estimated visually from the figures. If a particular rule is noise-resistant, the conditional transition probabilities will peak along the diagonal of the transition matrix. This will be indicated by large matrix element spikes along the diagonal of the X-

Y axis, shown. In general, we found that nearest-neighbor coupled cellular automata are weakly fault tolerant. Rules that promote large numbers of attractors tend to intertwine attractor basins. As a result, even small deviations in state from most attractor trajectories will cause a jump to another attractor.

Rules 25, 45, 94, and 108 show a slow increase in their I_{SR} parameters with array size. Rule 25 is composed primarily of attractor lengths which are integral multiples of array size. As array size increases, more evenly divisible cycle lengths are permitted. This leads to an overall increase in the information required to predict a given attractor. Rule 108 displays the inverse behavior of the monotonic rules discussed above. For this rule, the two possible attractors tend to share equally the exponentially increasing volume of state space as array size grows. Finally, Rule 45 is the most complex rule observed within the class. This rule is characterized by a set of uncorrelated attractor sizes, some of which are nearly fully ergodic. It is not possible to explain simply the behavior of this rule in general terms.



Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F1. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

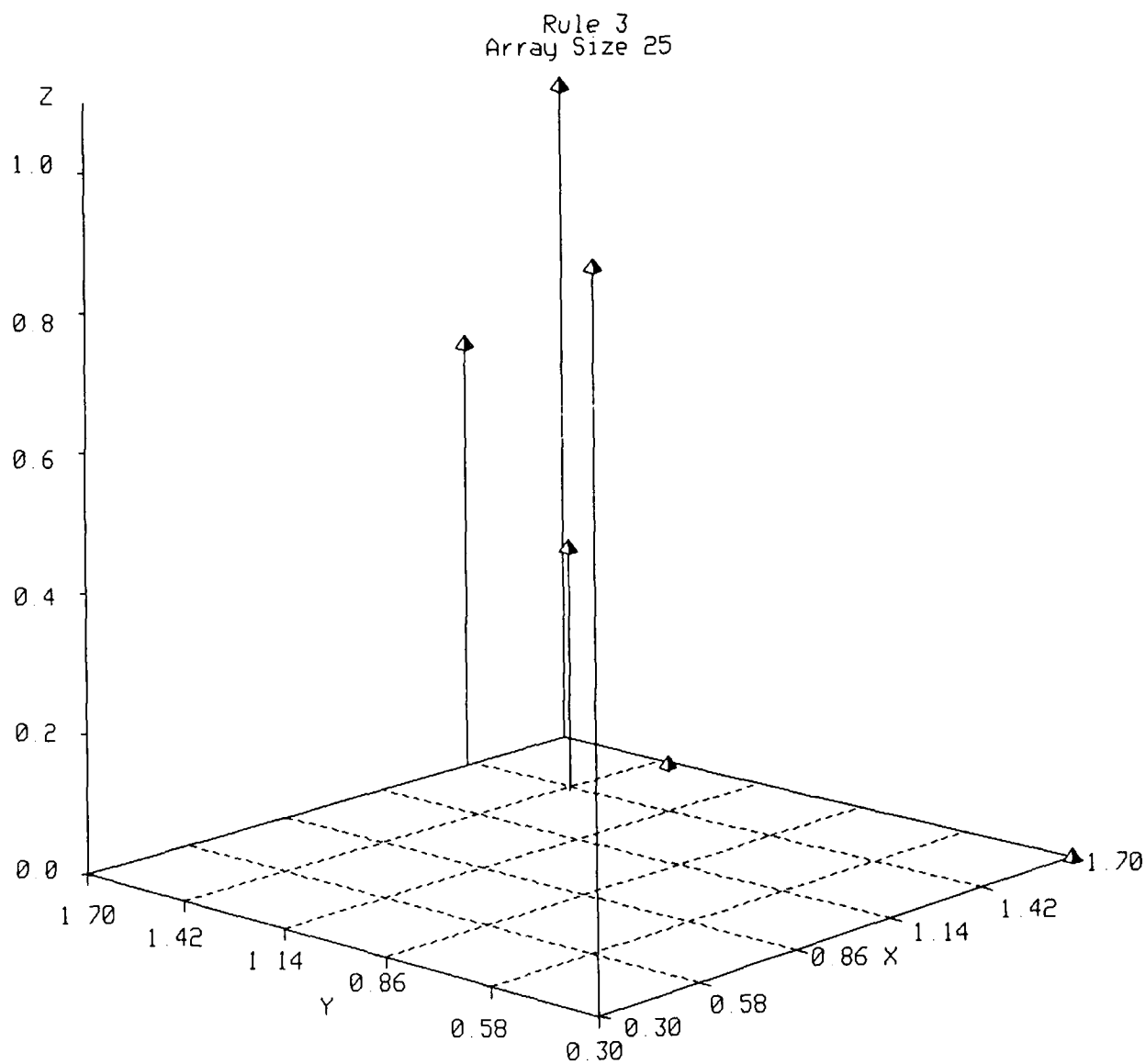


Figure F2. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

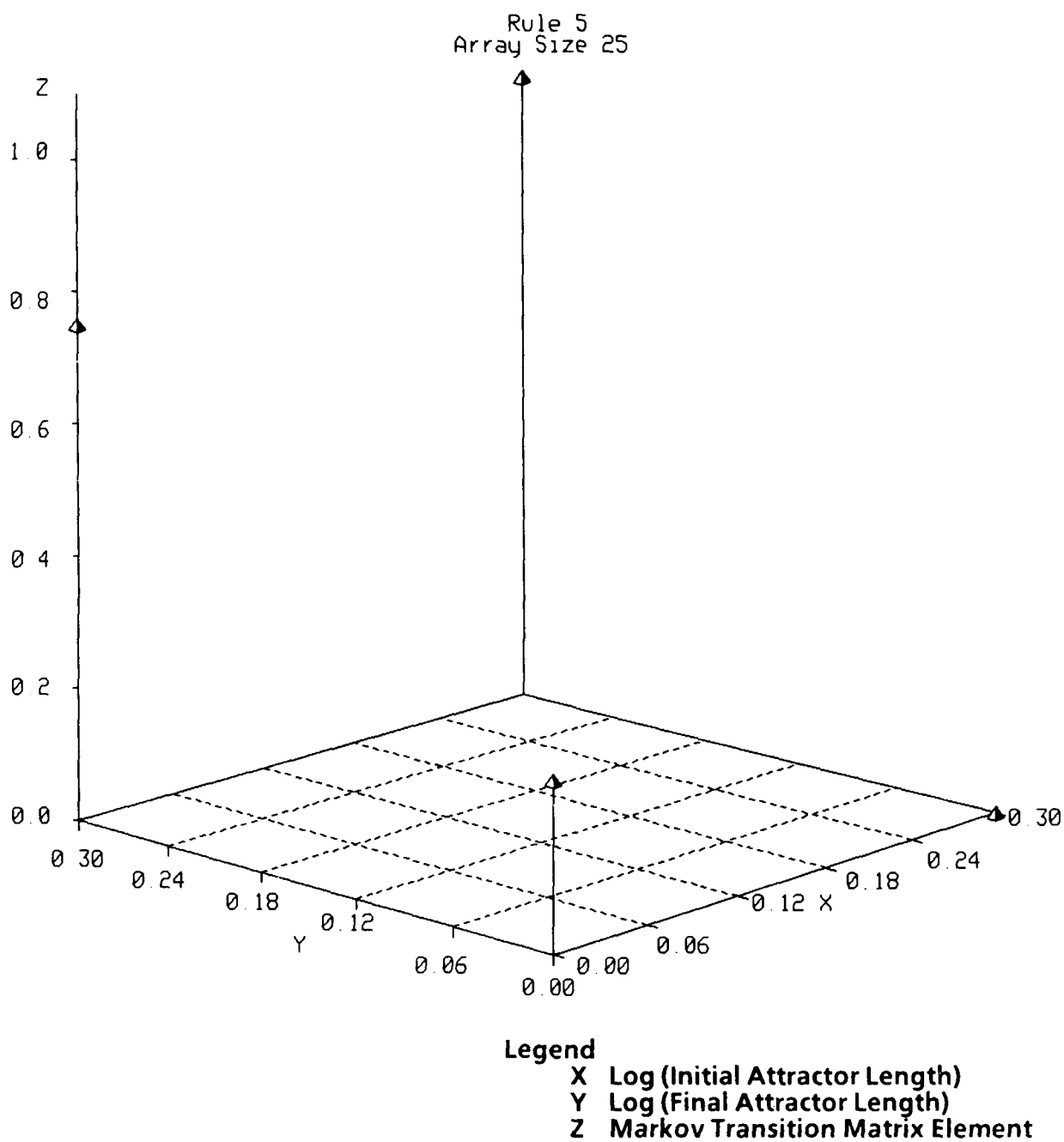


Figure F3. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

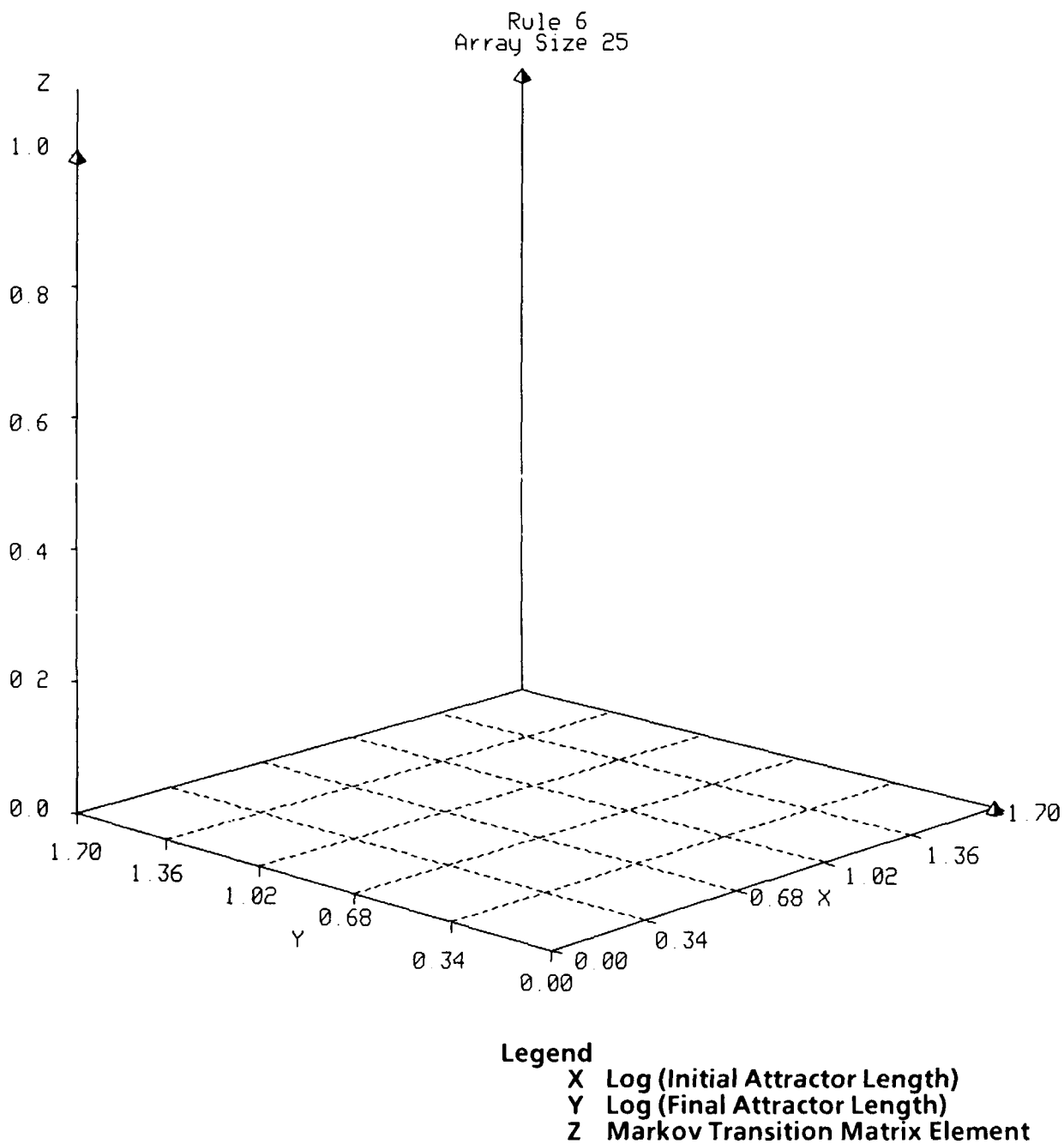


Figure F4. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

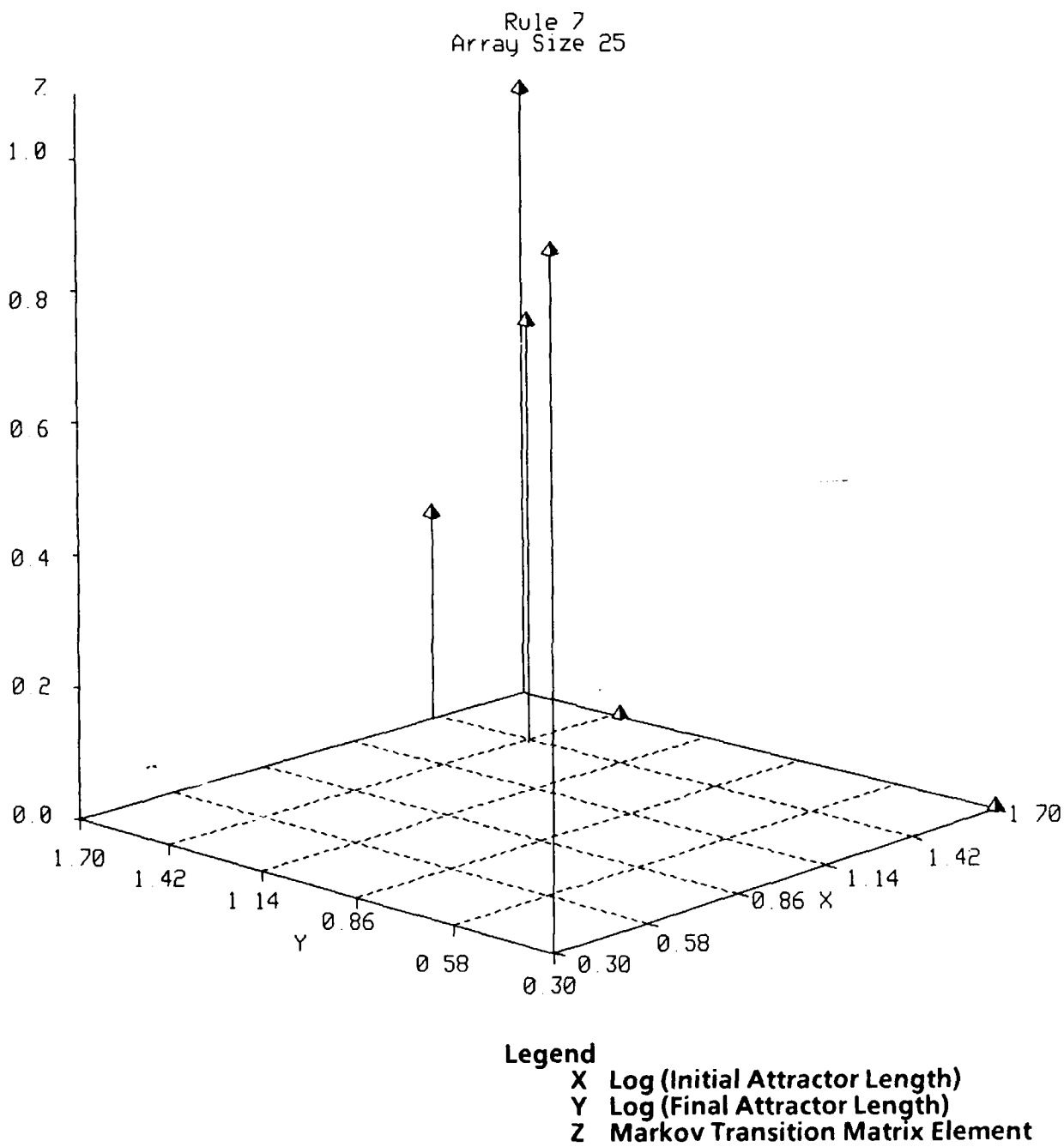
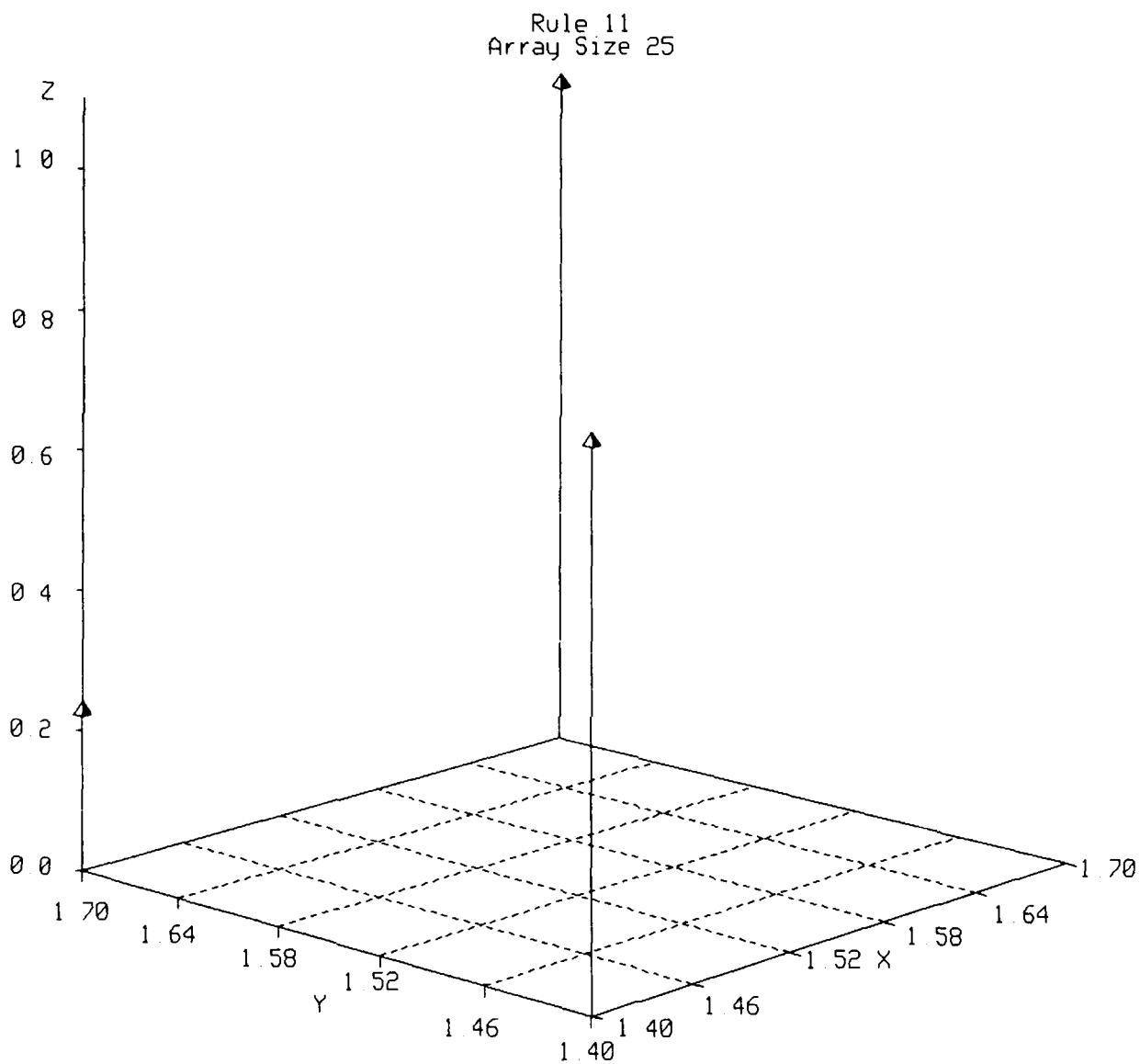


Figure F5. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

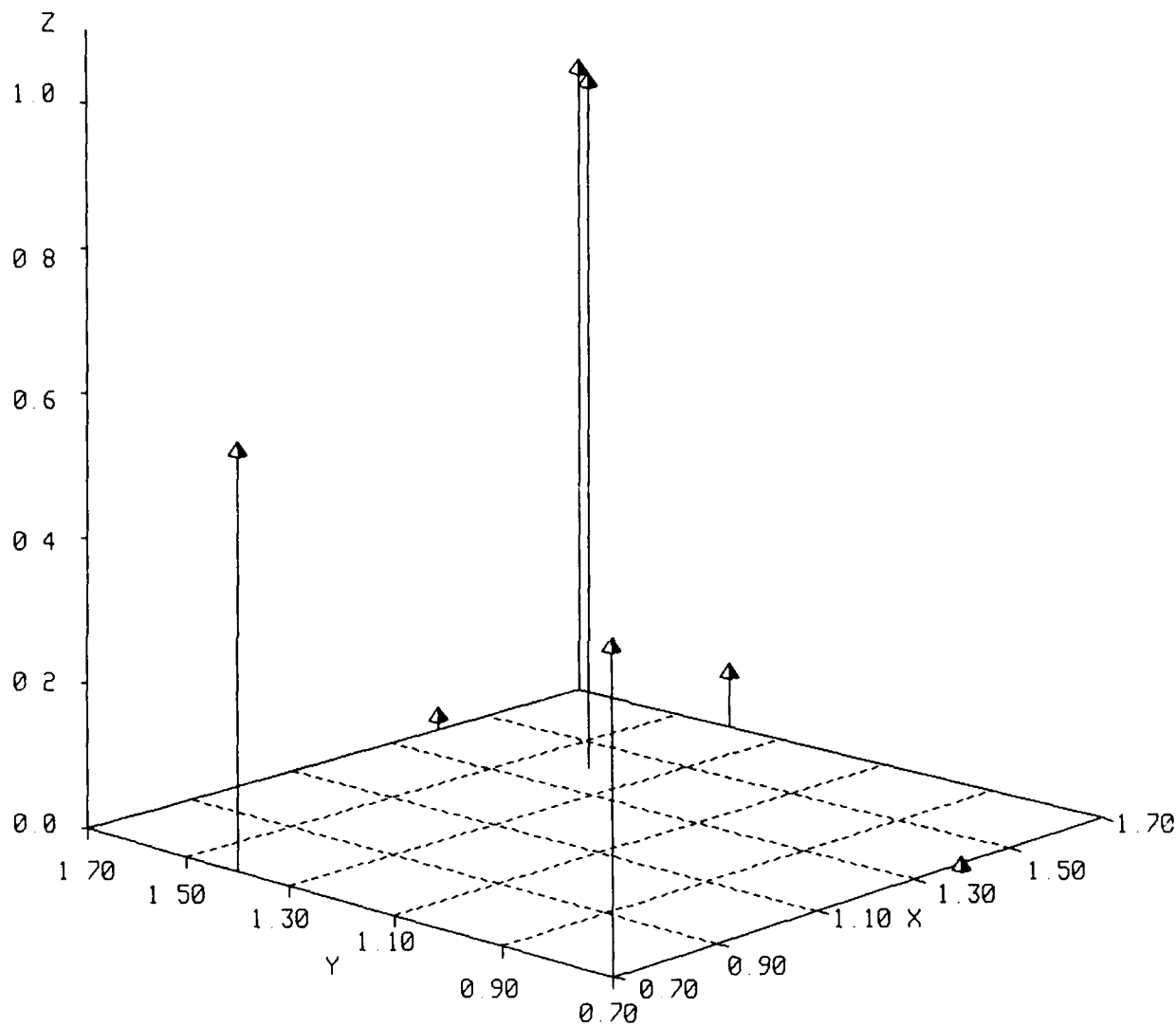


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F6. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 14
Array Size 25

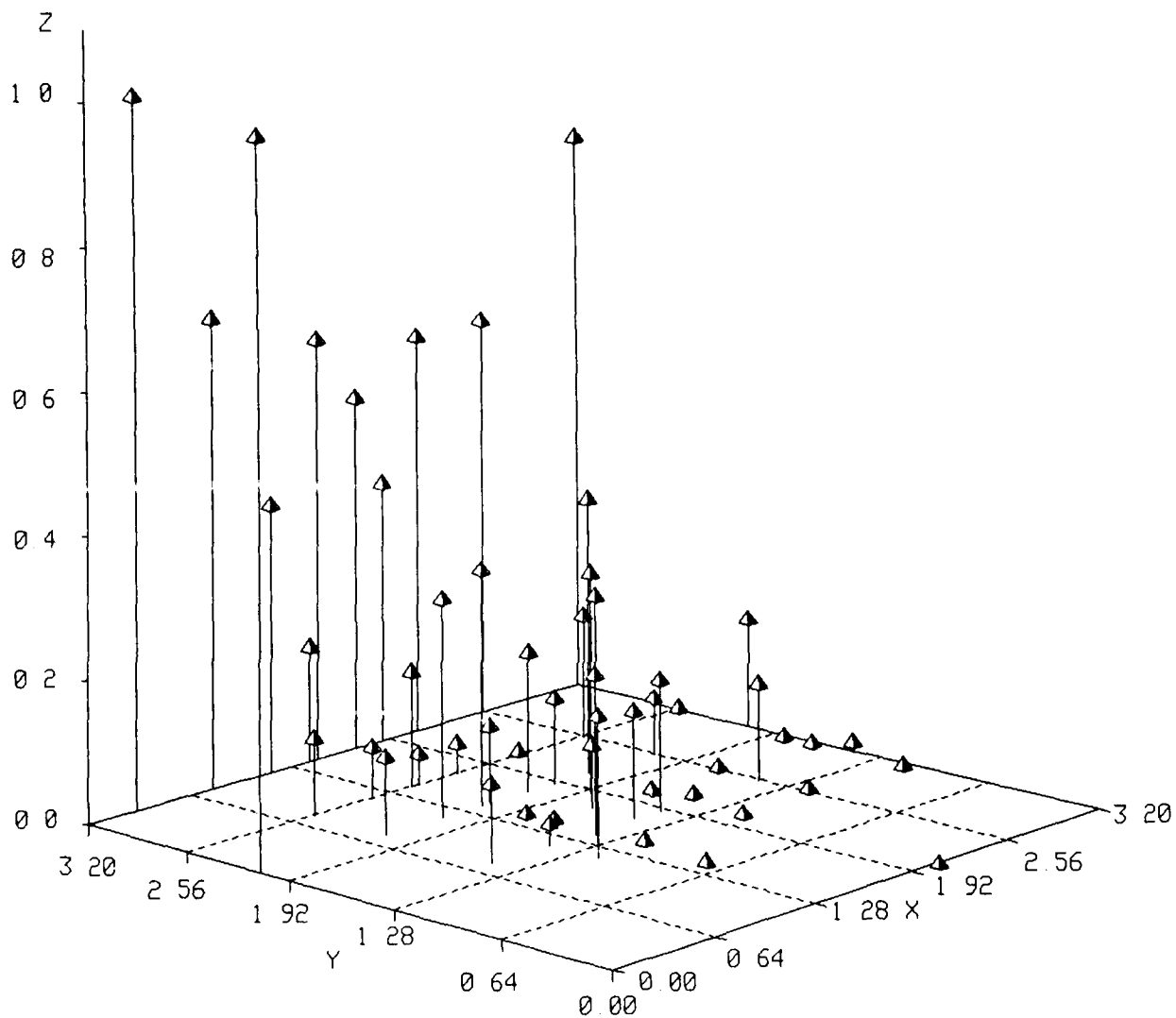


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F7. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 18
Array Size 25

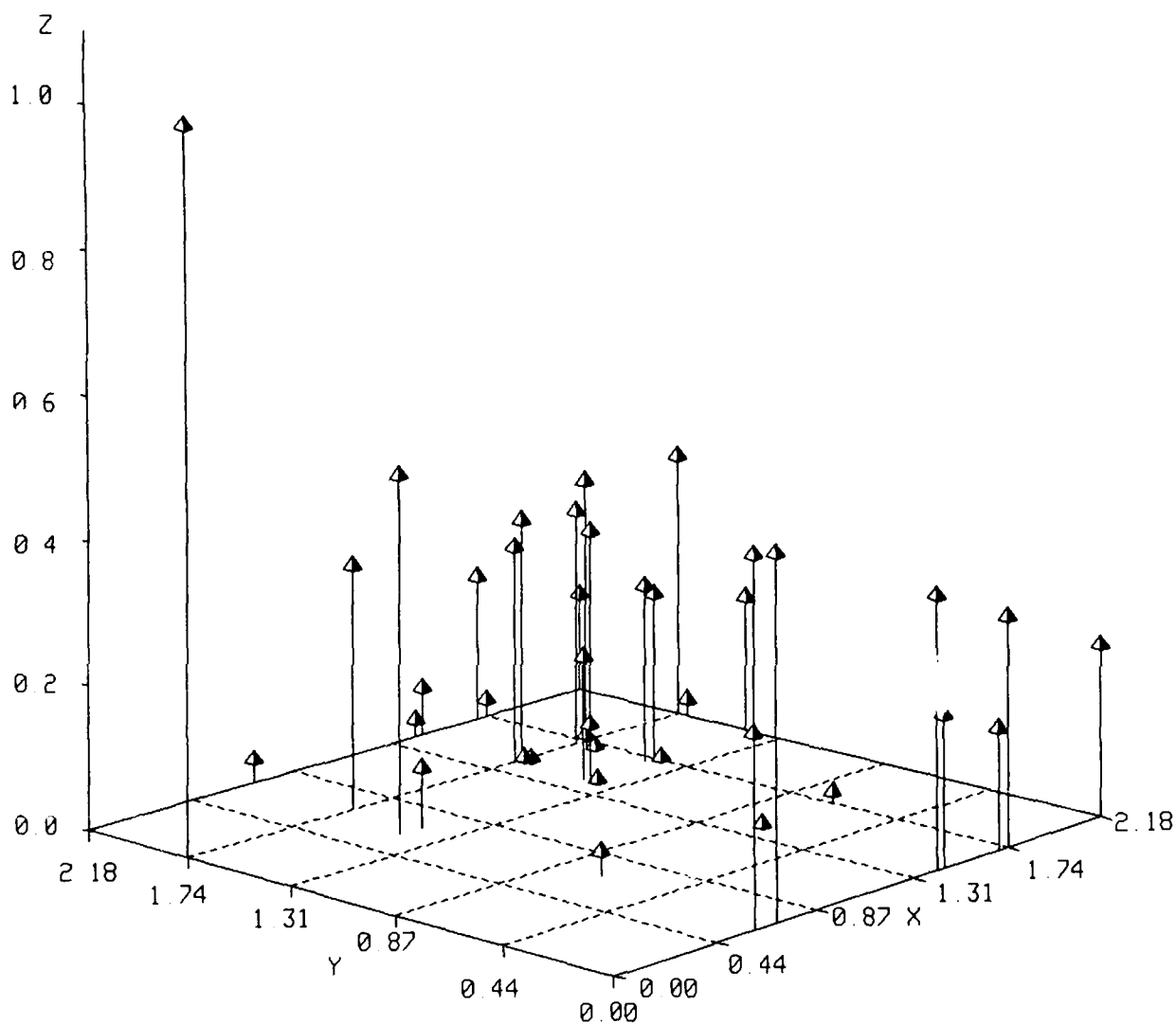


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F8. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 22
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F9. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

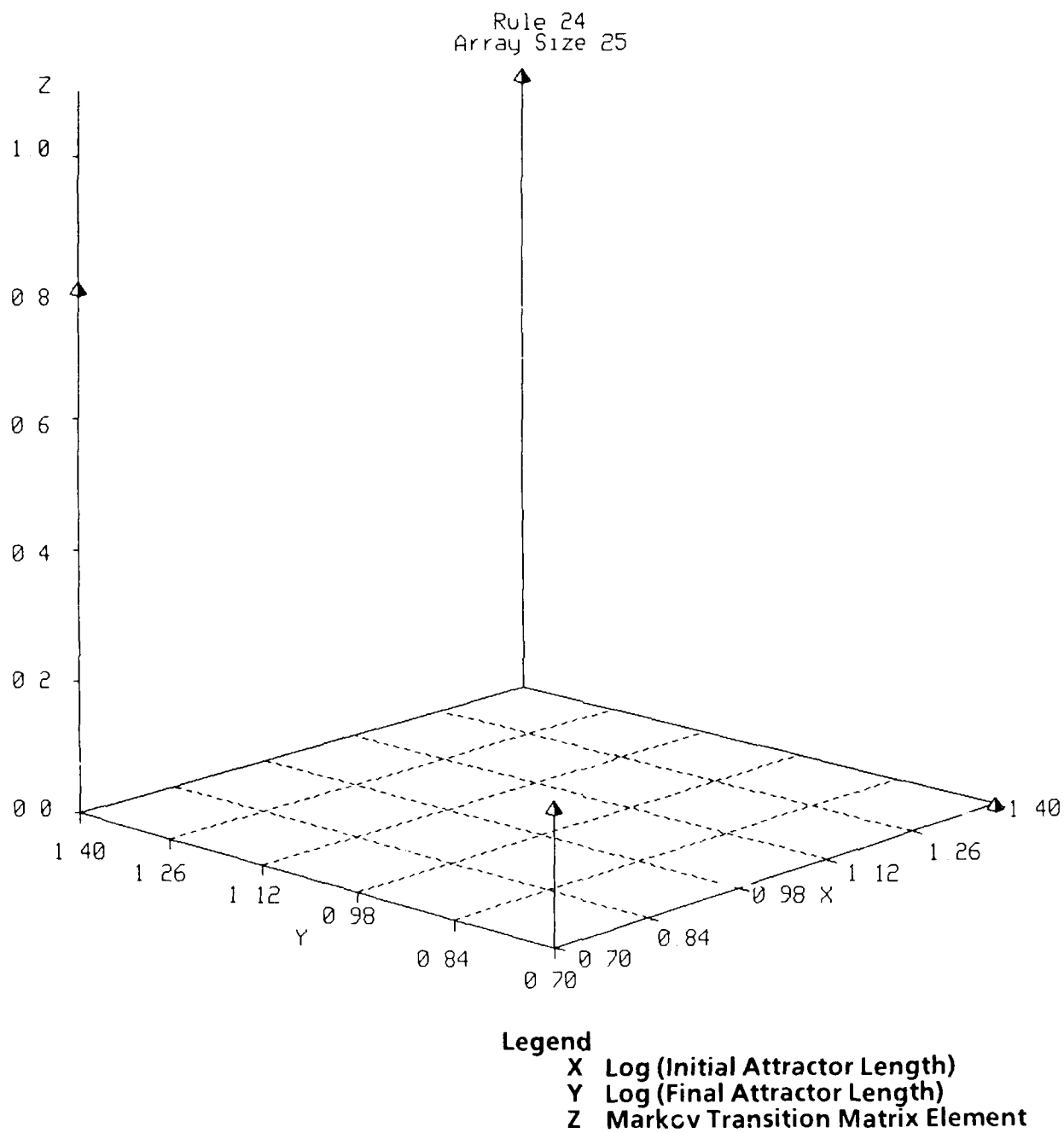
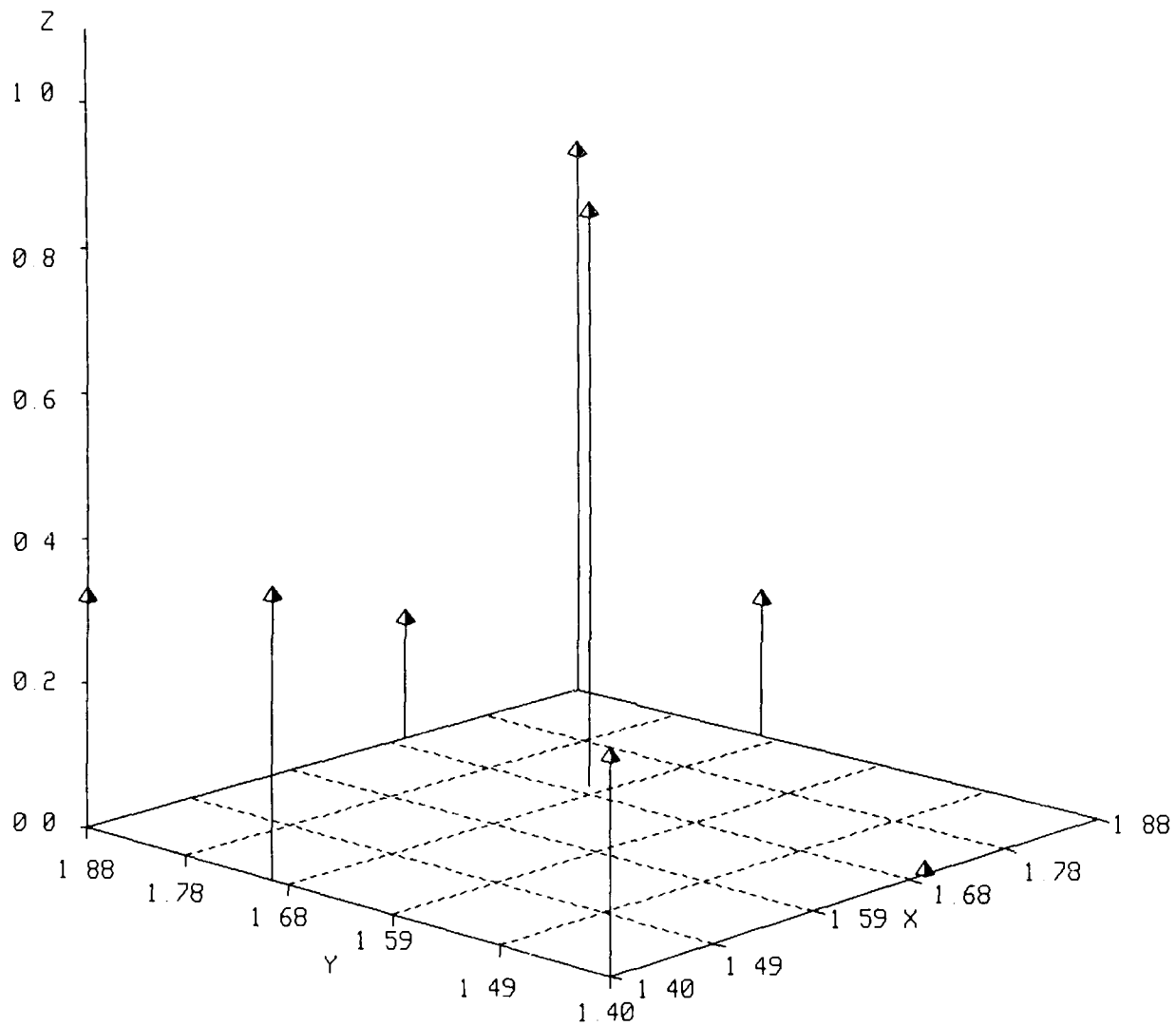


Figure F10. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 25
Array Size 25



Legend

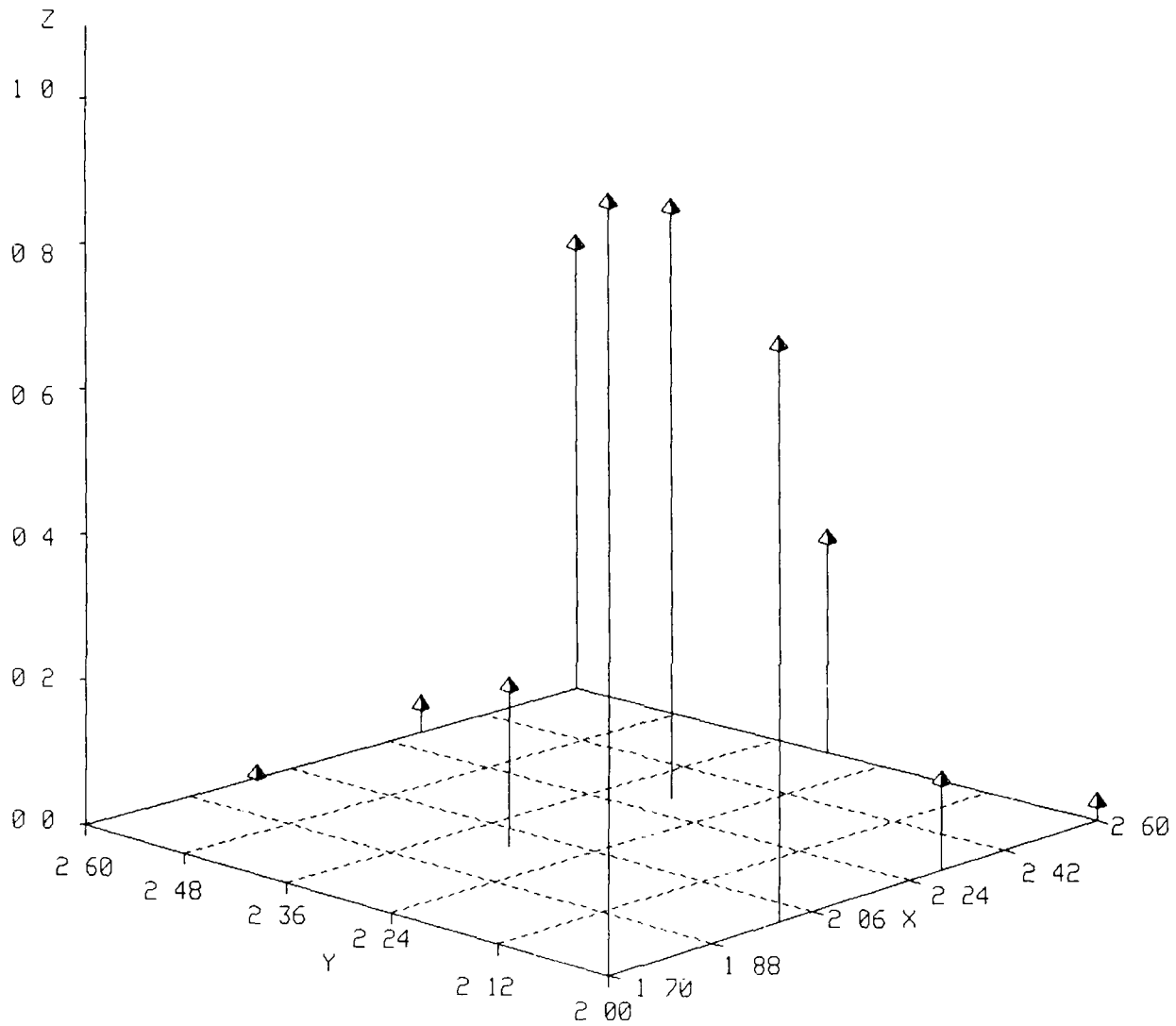
X Log (Initial Attractor Length)

Y Log (Final Attractor Length)

2 Markov Transition Matrix Element

Figure F11. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 26
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F12. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

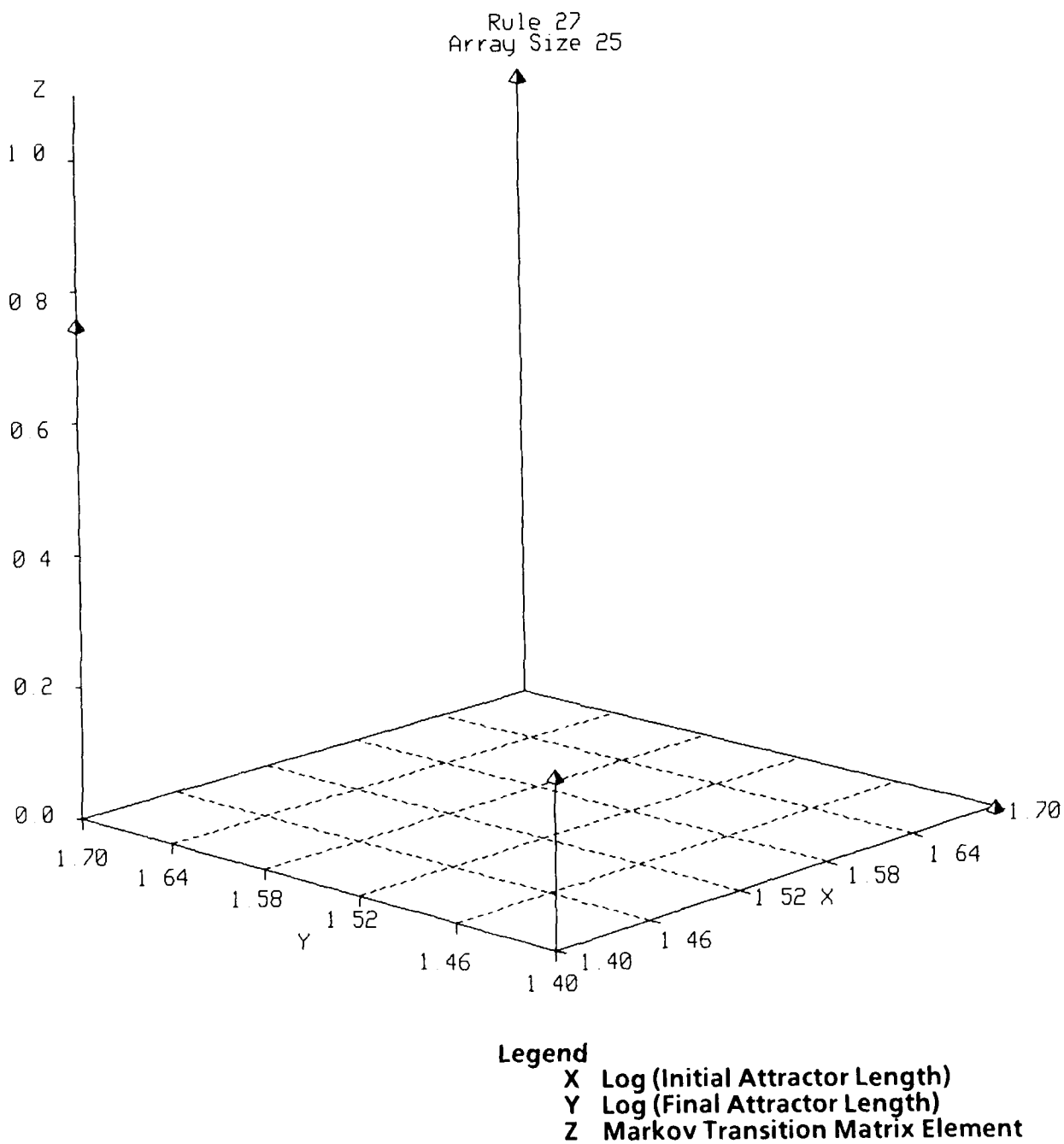


Figure F13. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

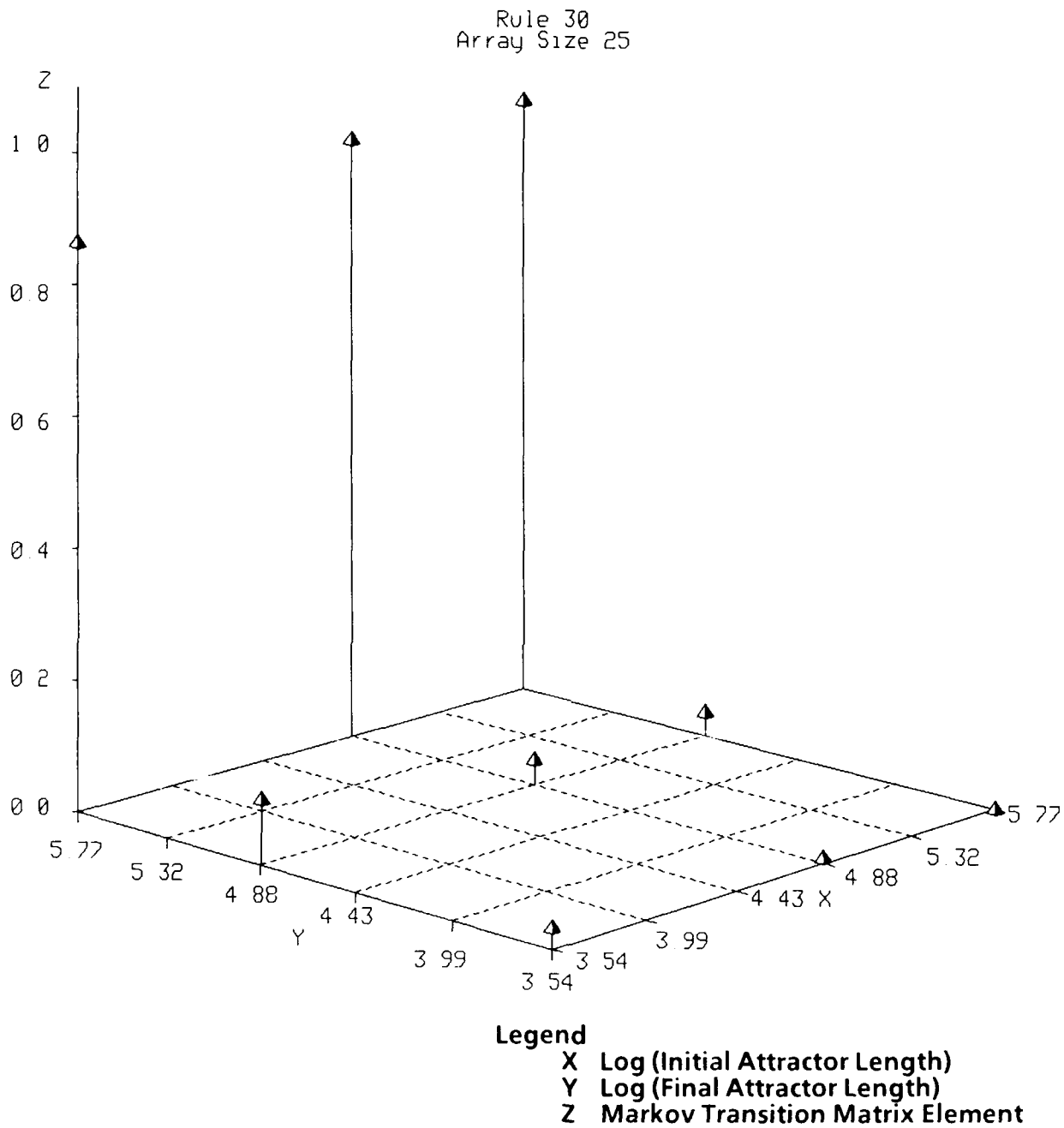


Figure F14. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

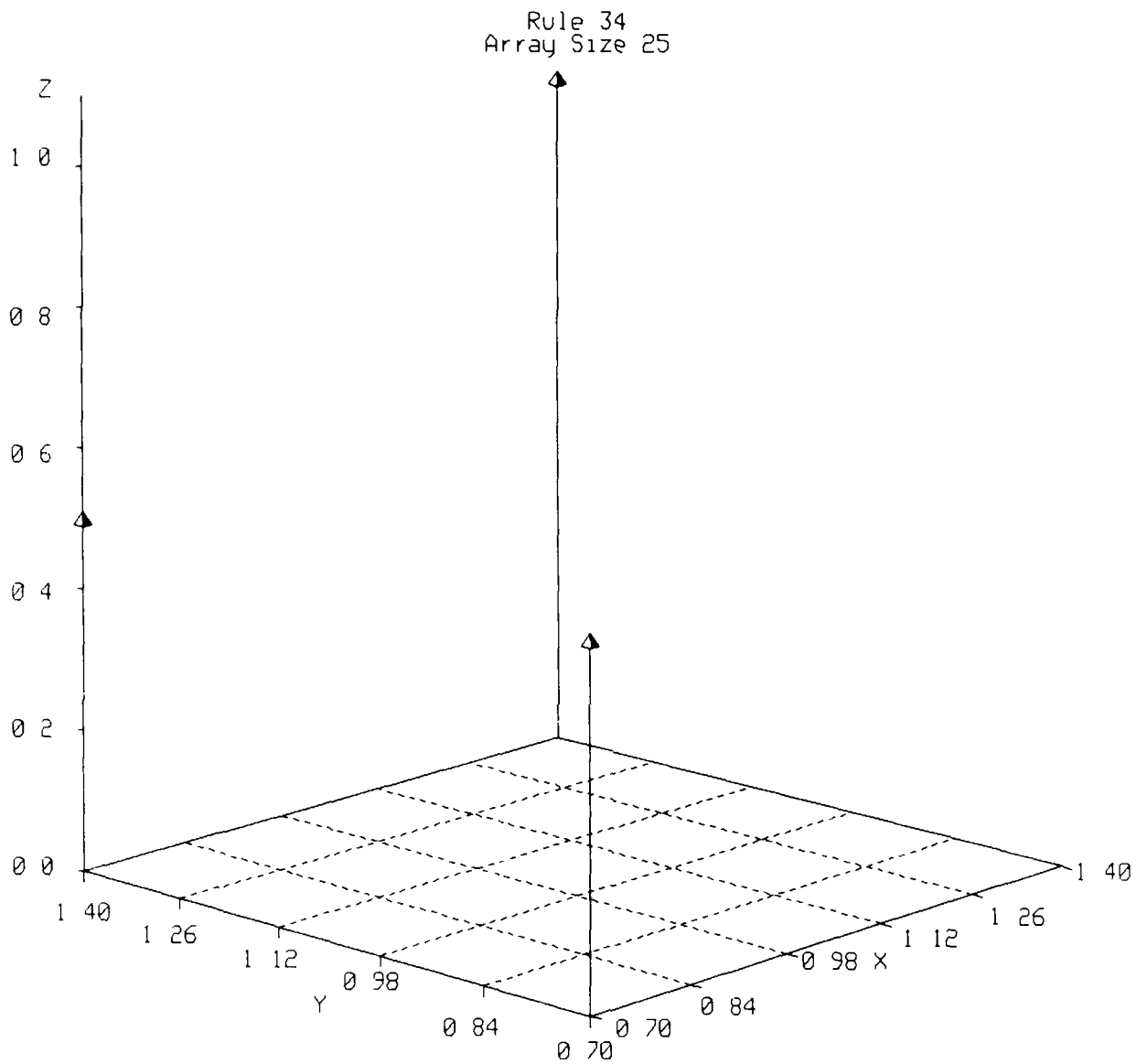


Figure F15. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

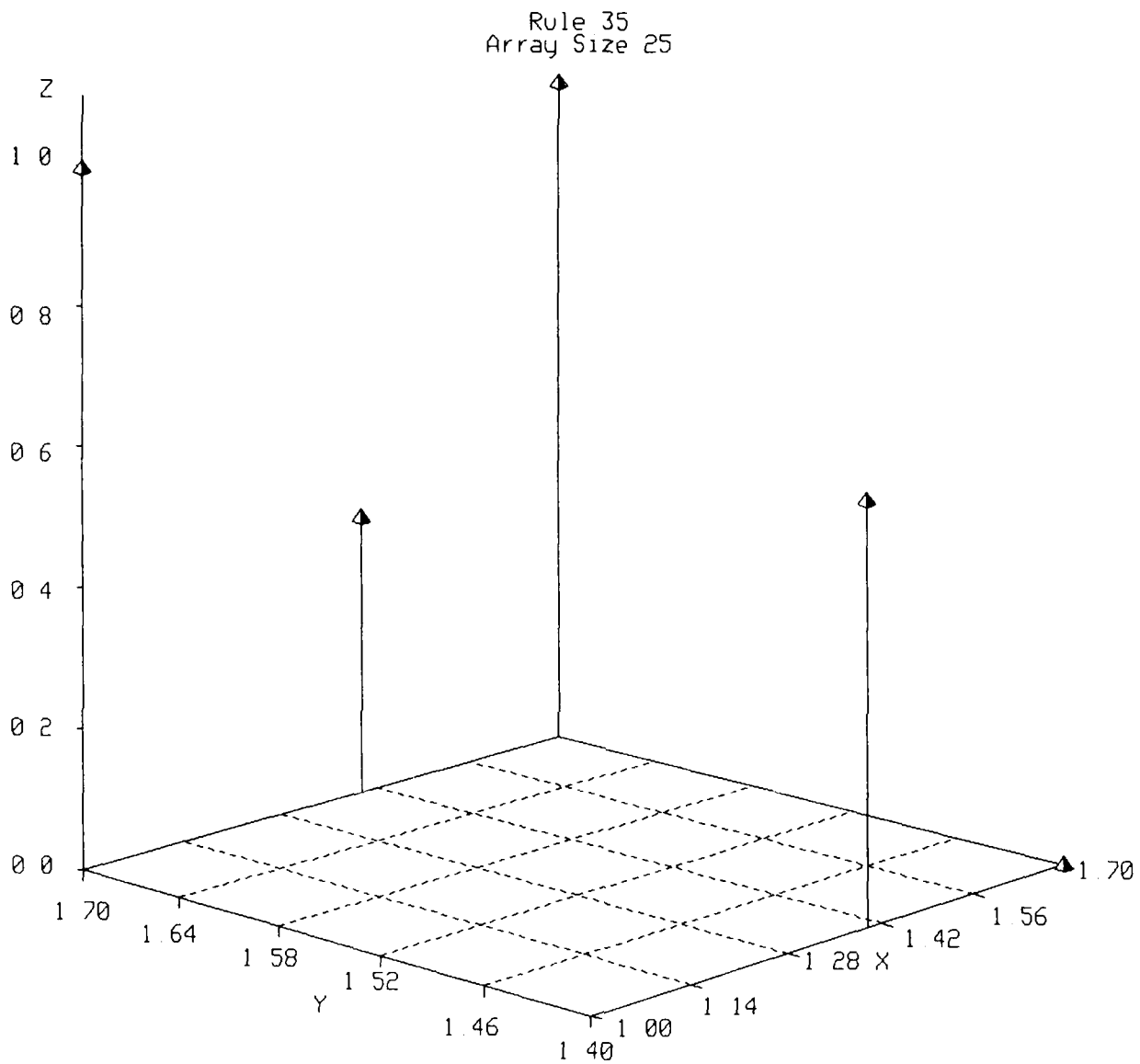
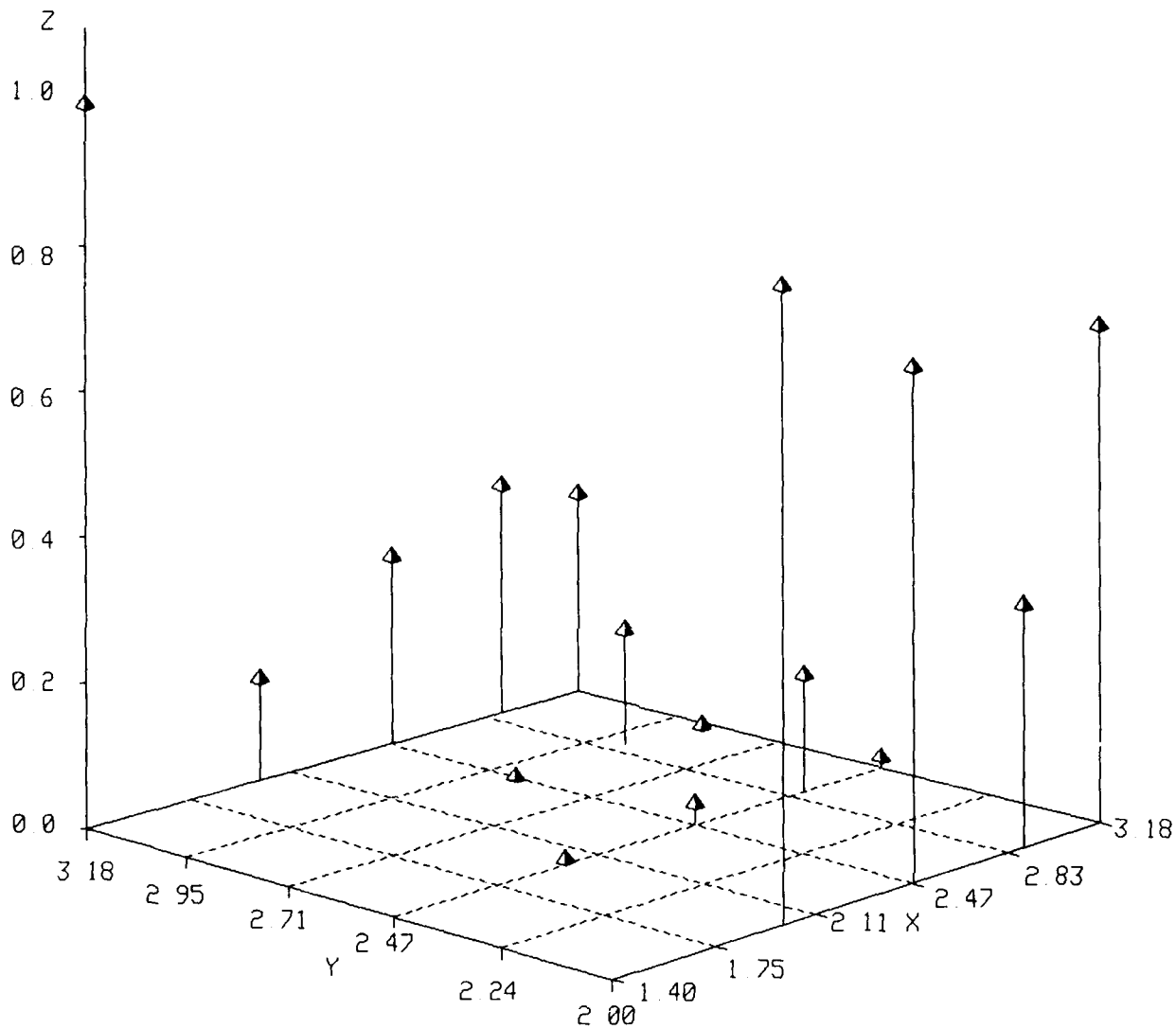


Figure F16. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

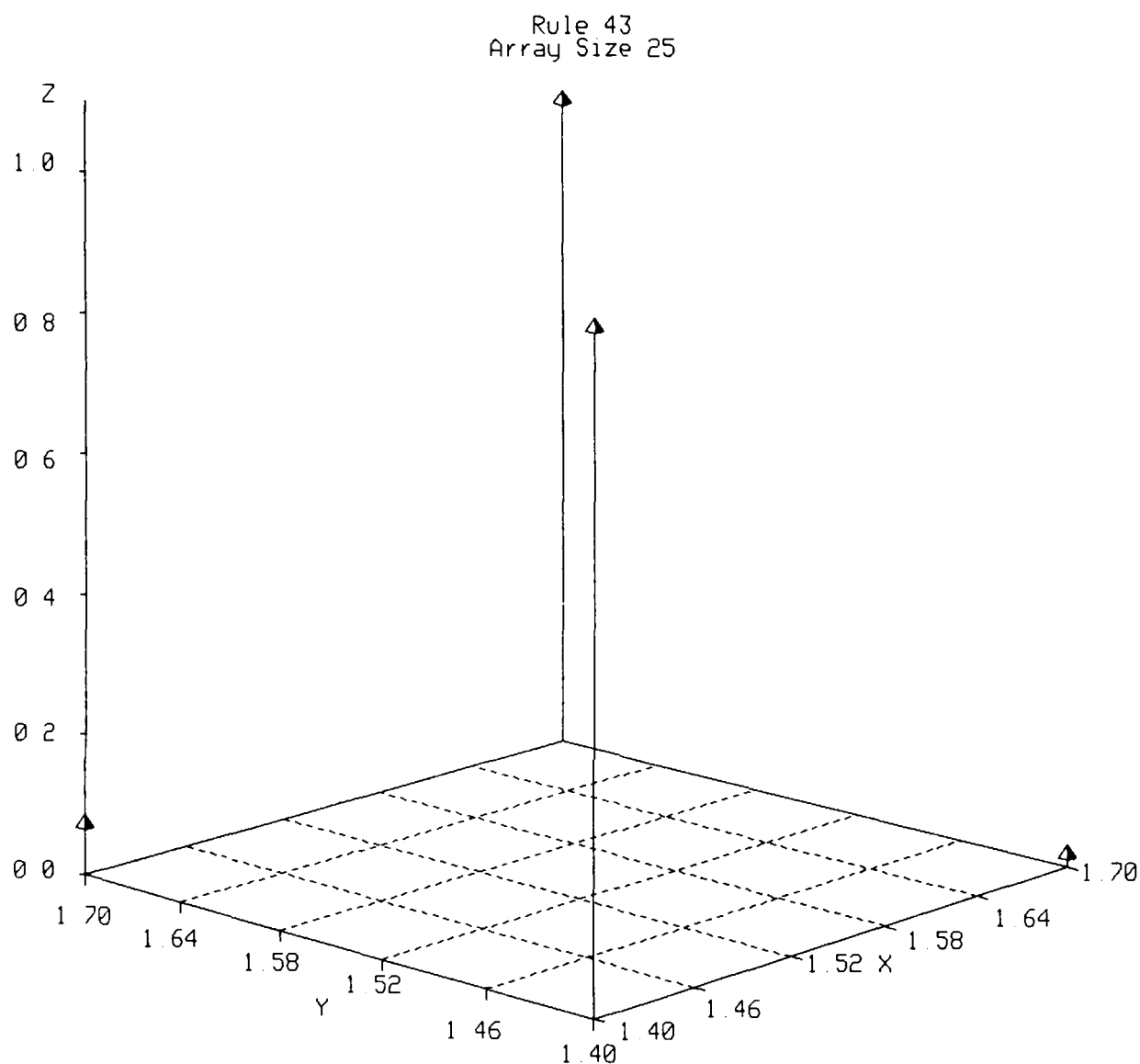
Rule 41
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F17. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

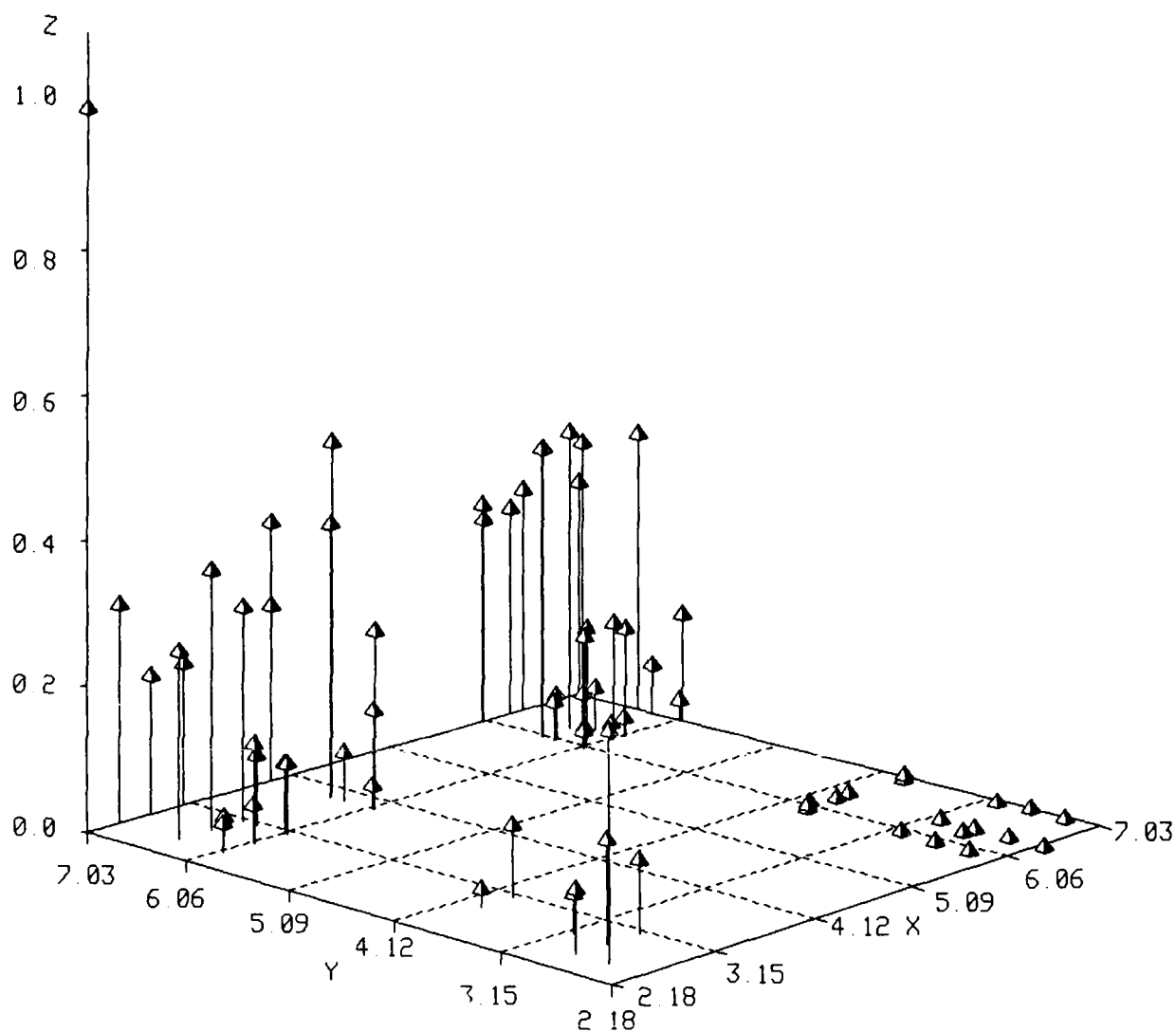


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F18. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

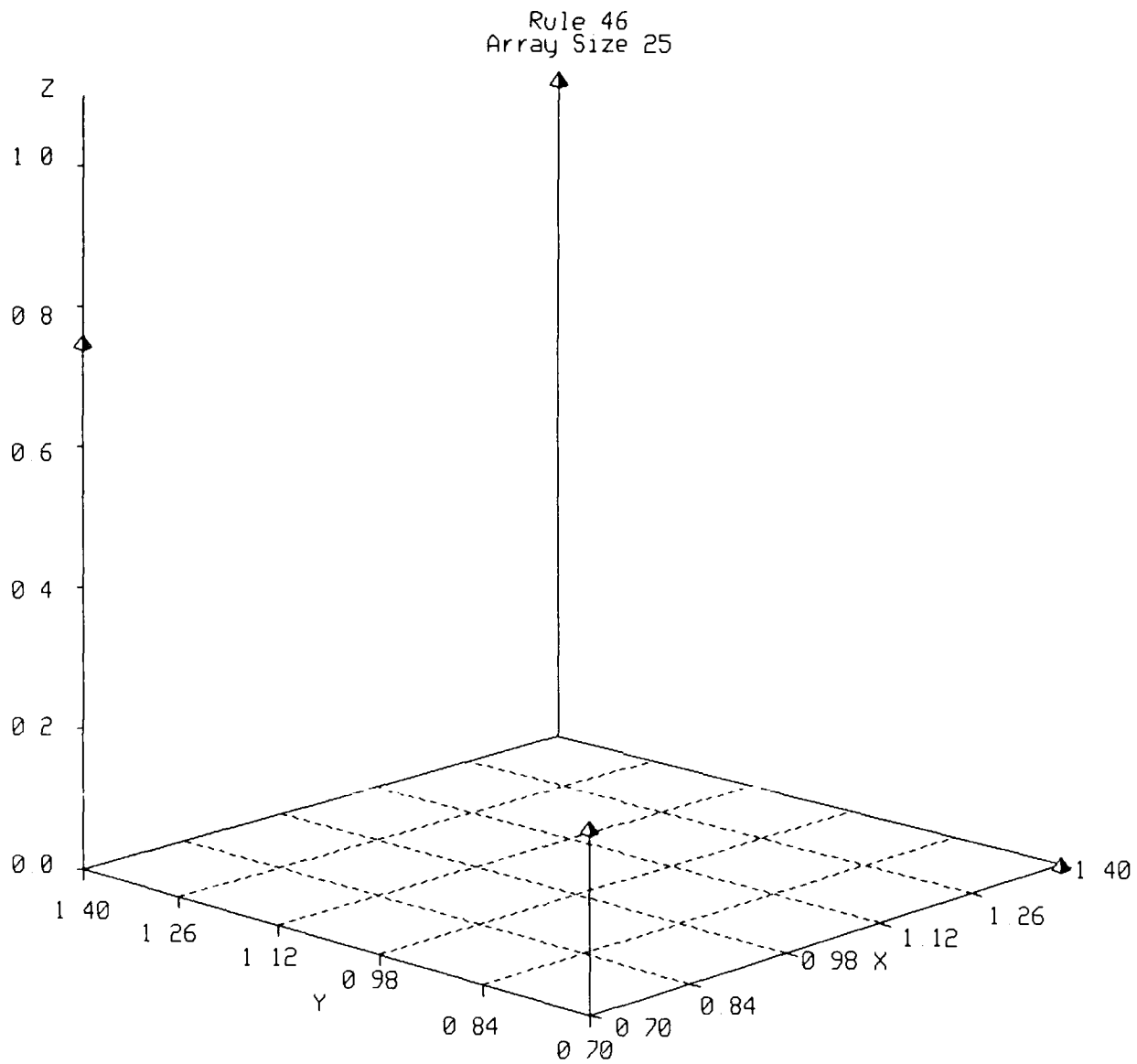
Rule 45
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F19. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

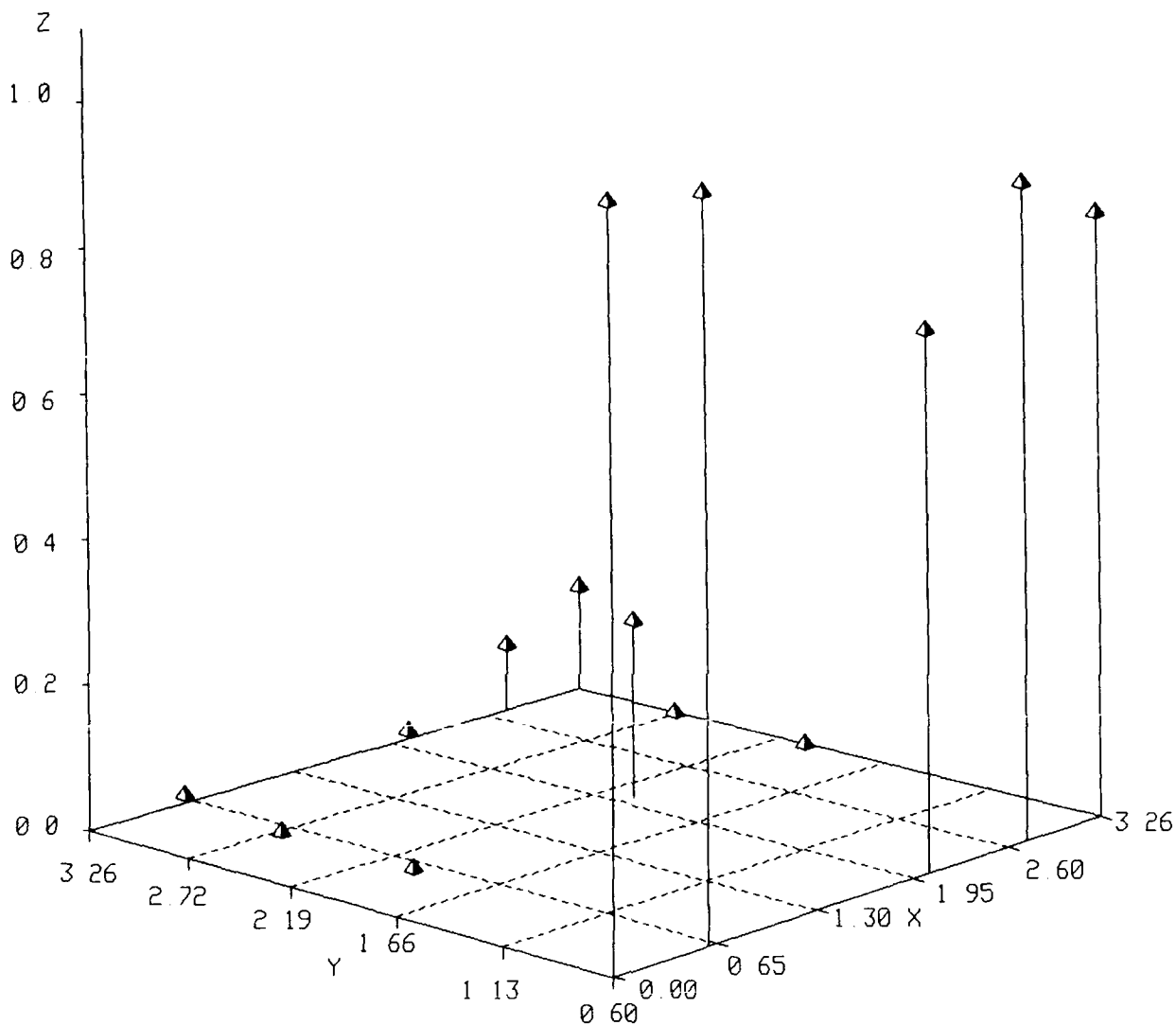


Legend

- X** Log (Initial Attractor Length)
- Y** Log (Final Attractor Length)
- Z** Markov Transition Matrix Element

Figure F20. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 54
Array Size 25

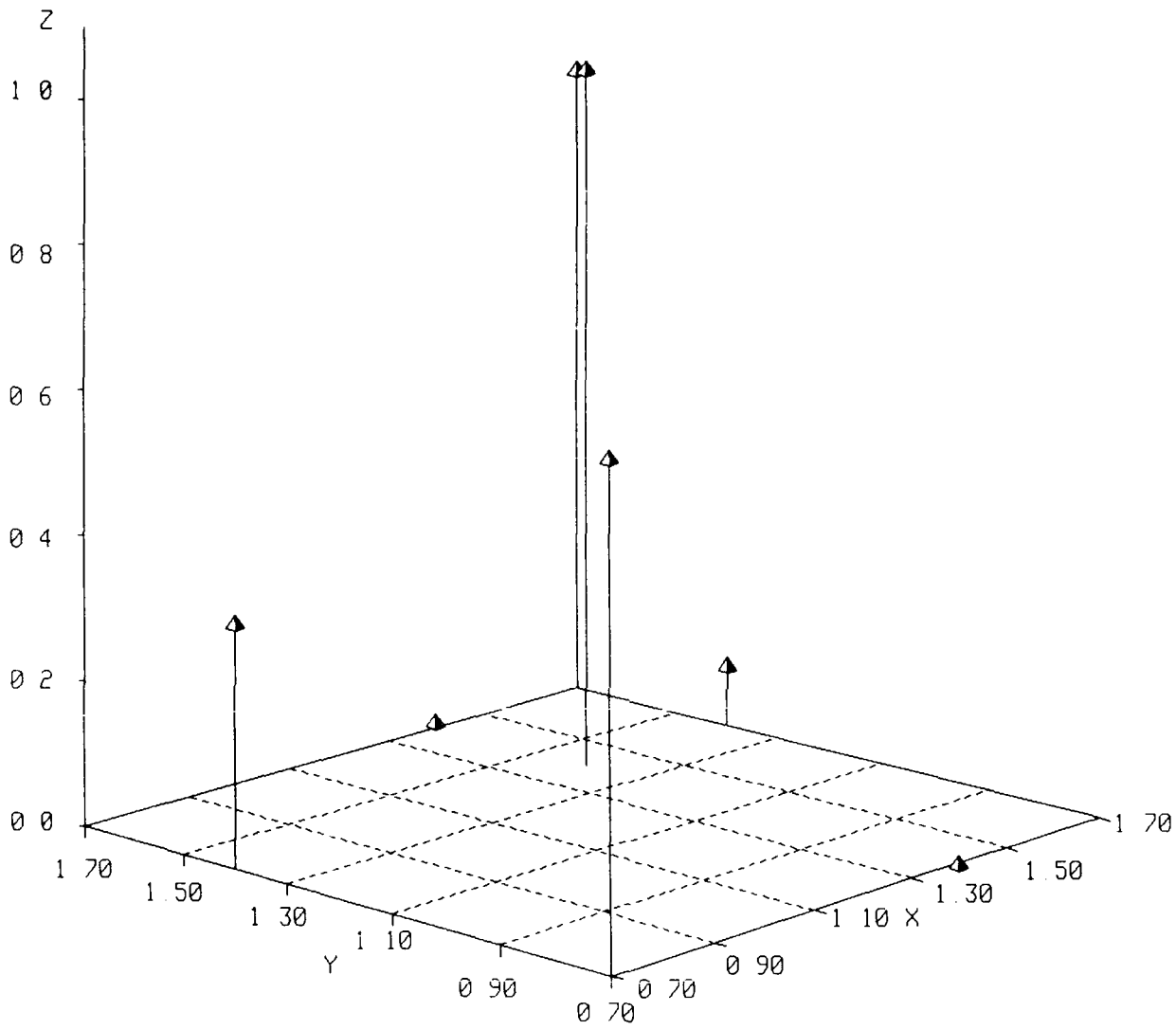


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F21. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 58
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F22. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

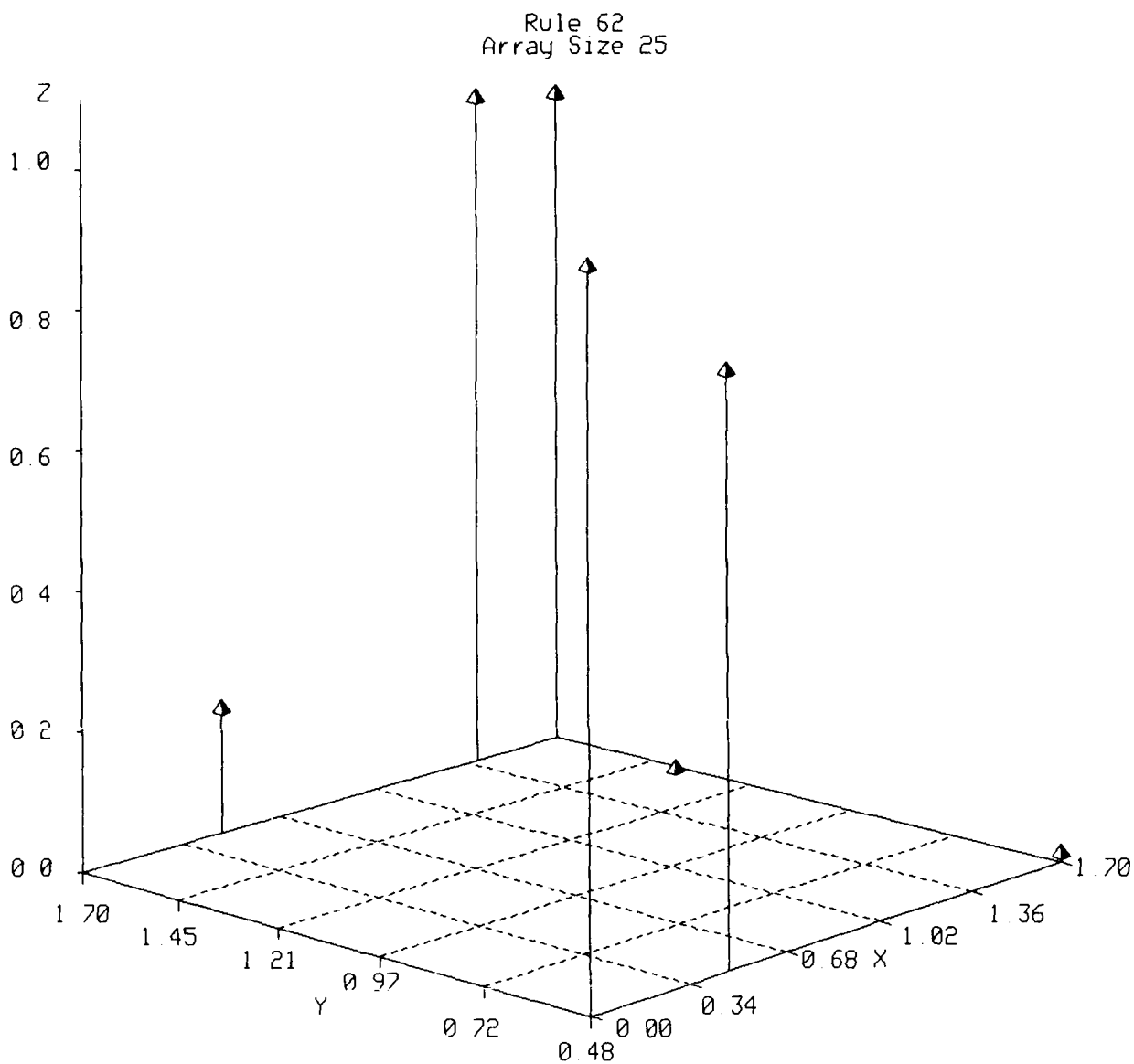
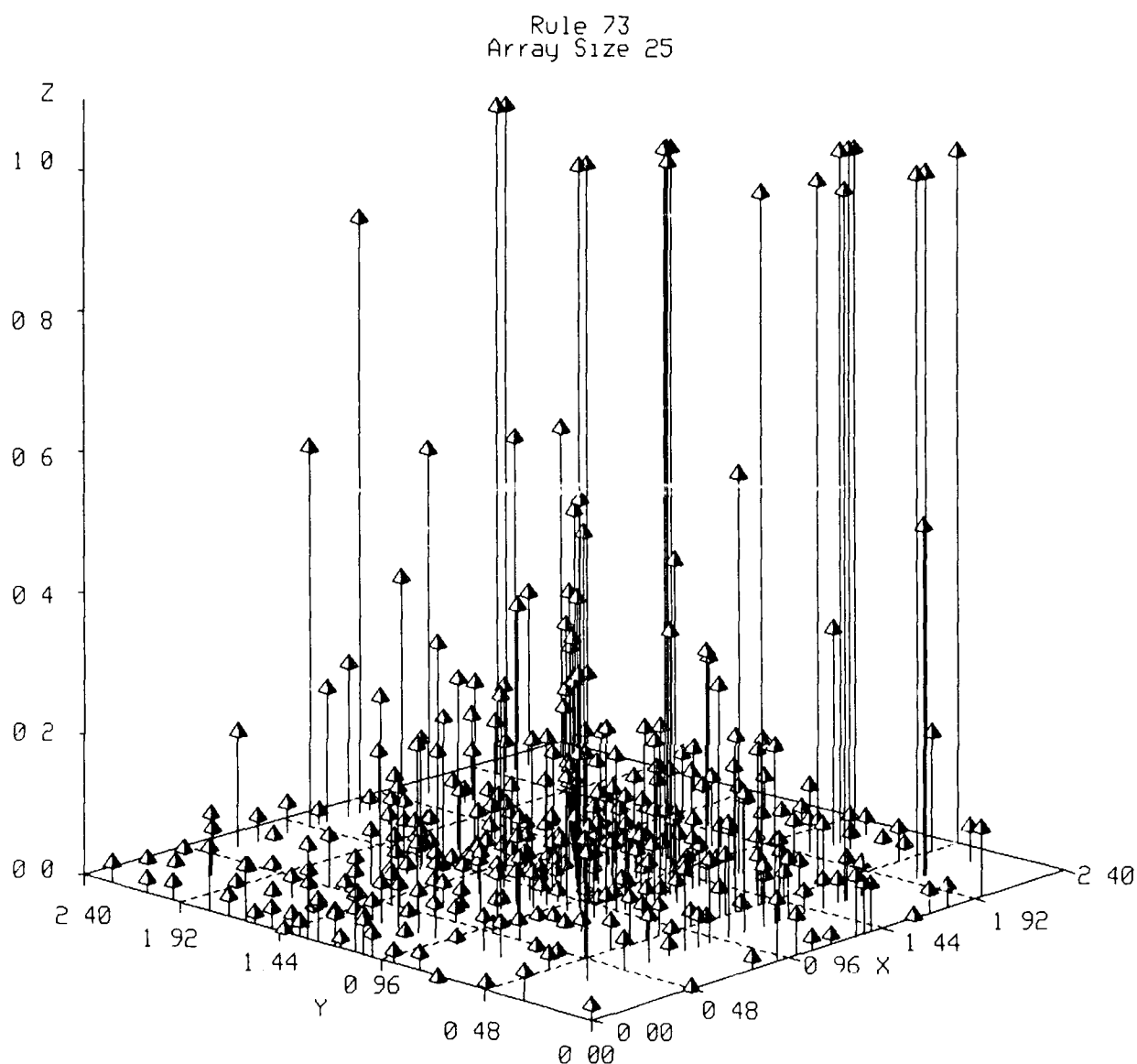


Figure F23. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

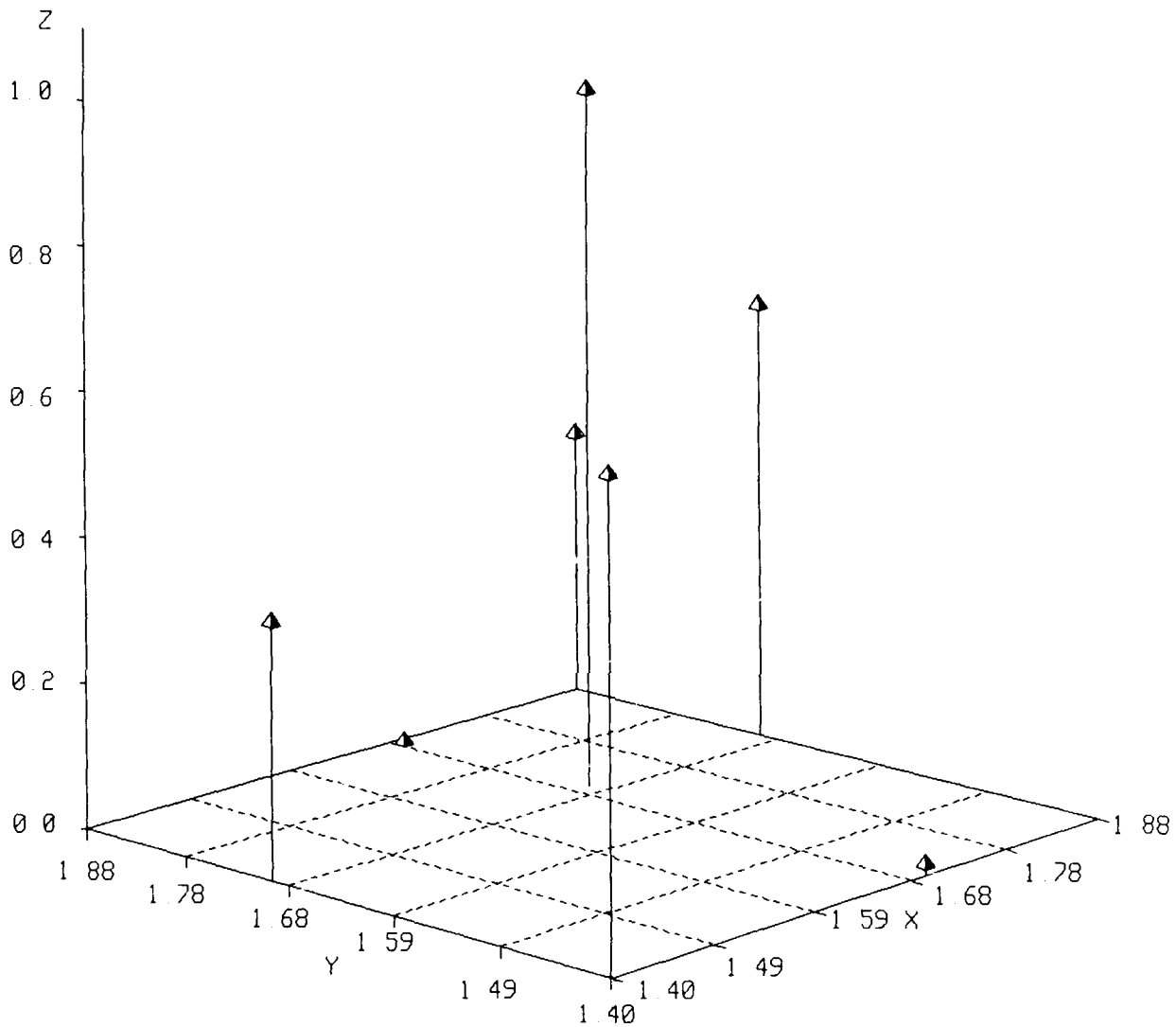


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F24. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 74
Array Size 25

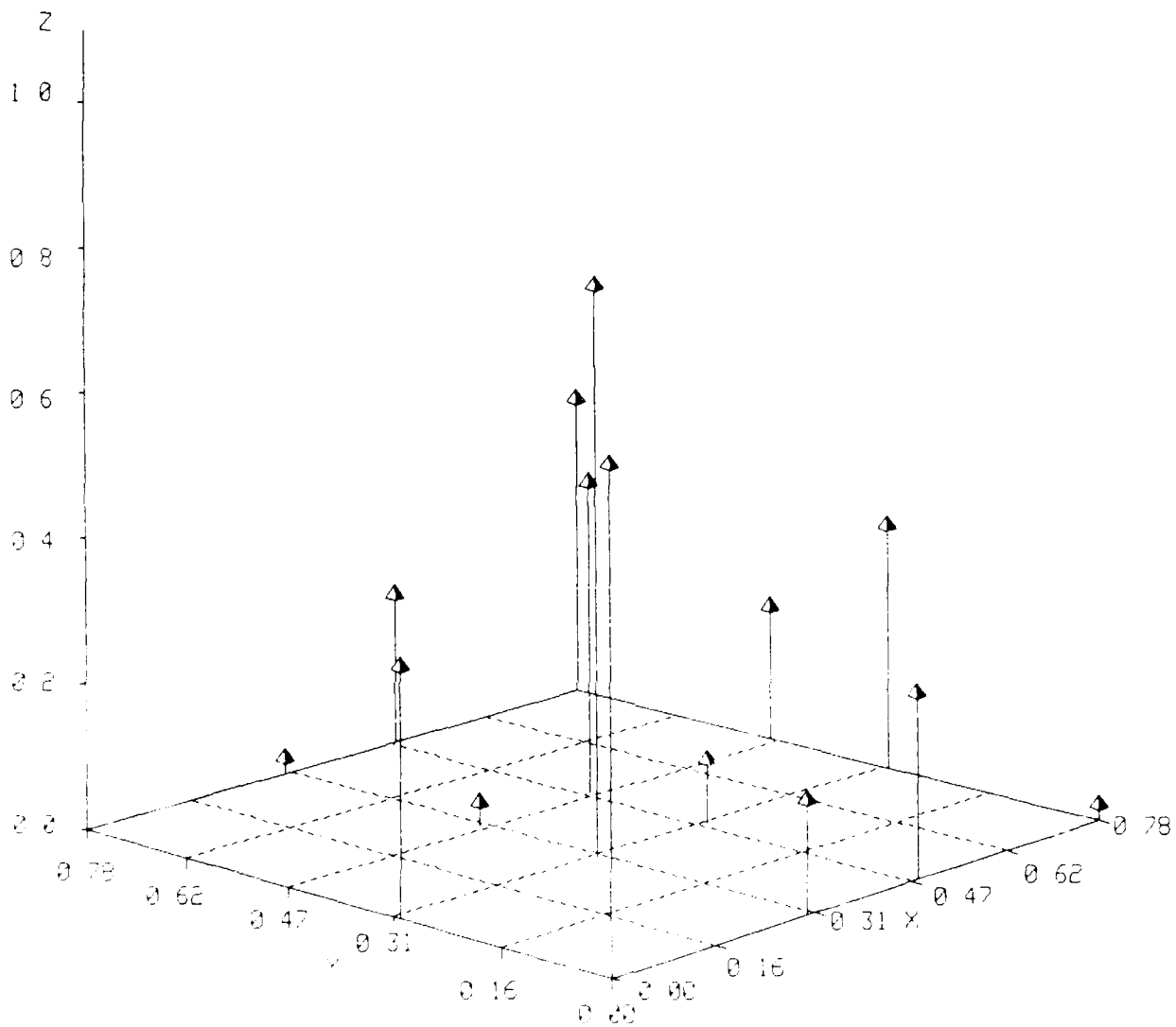


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F25. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 94
Array Size 25

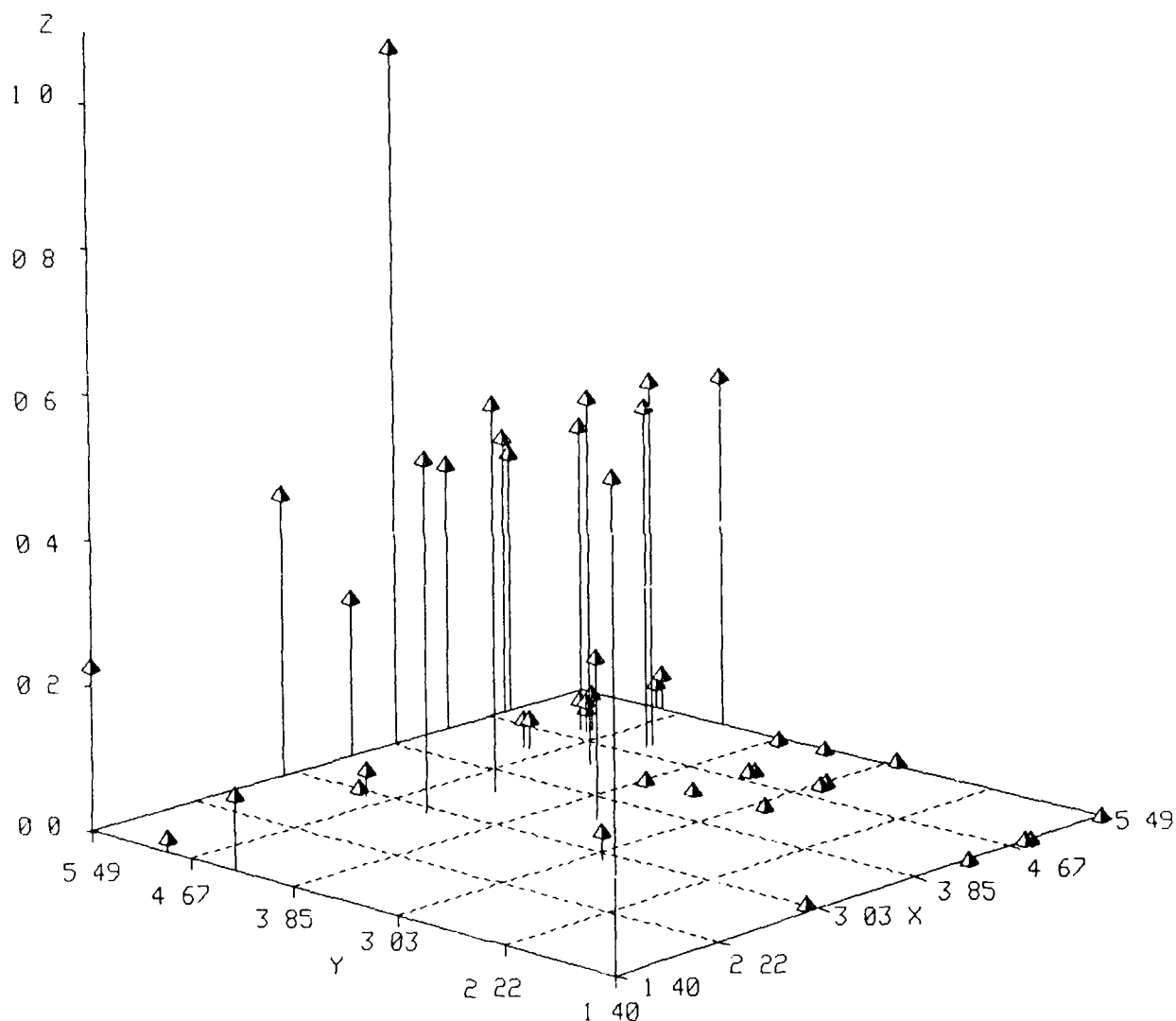


Legend

X Log (initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F26. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 106
Array Size 25

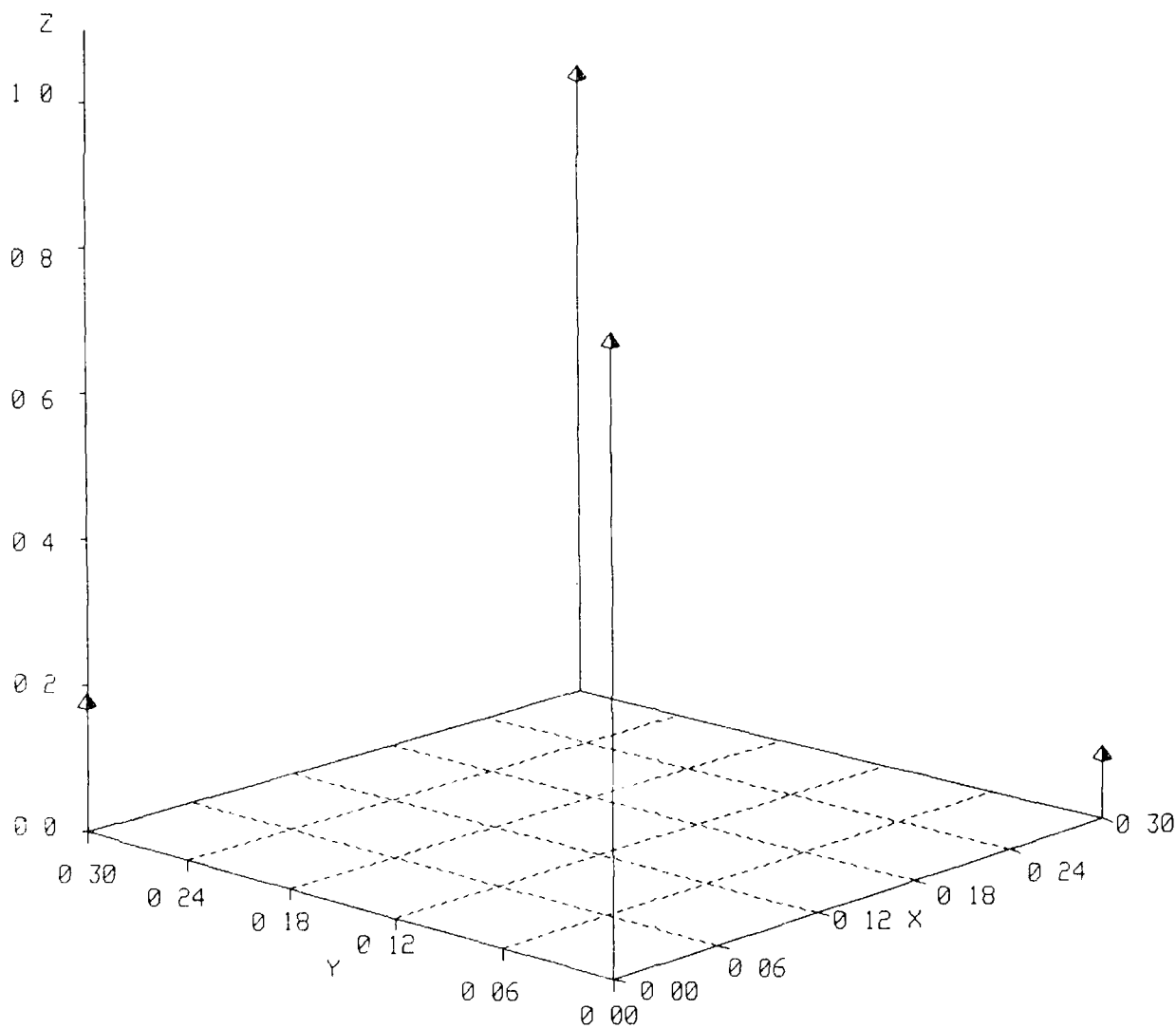


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F27. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

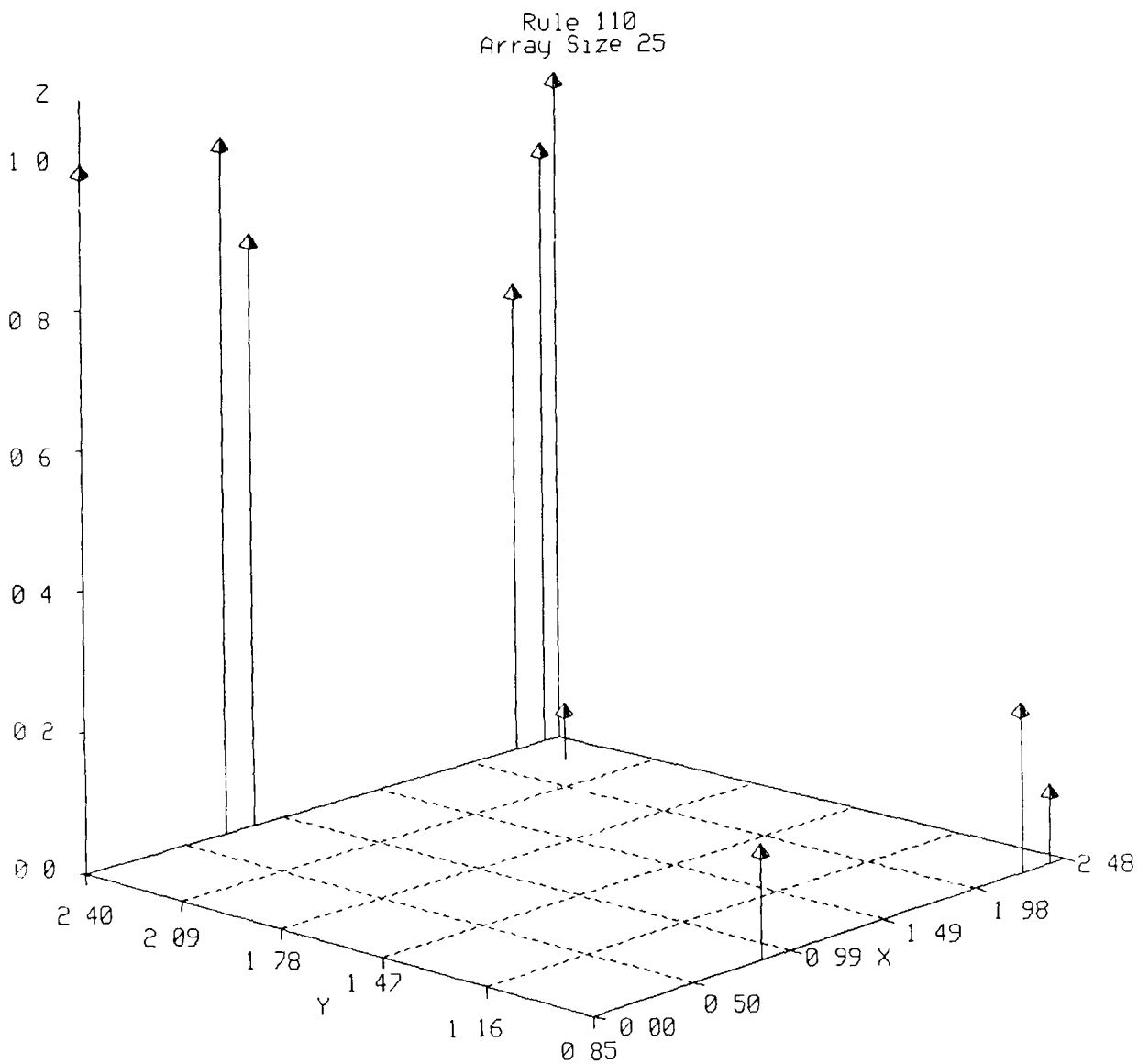
Rule 100
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F28. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

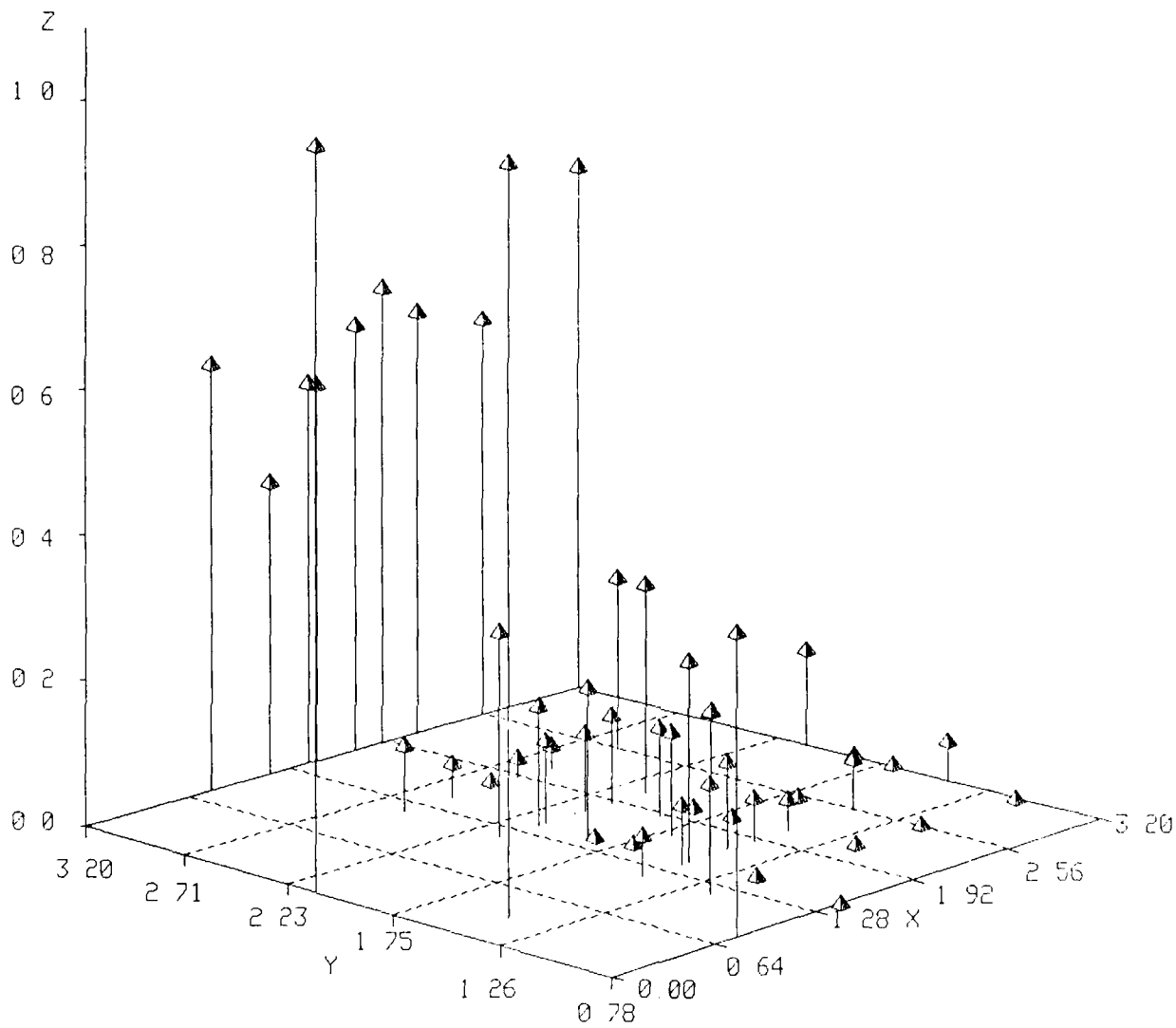


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F29. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 122
Array Size 25

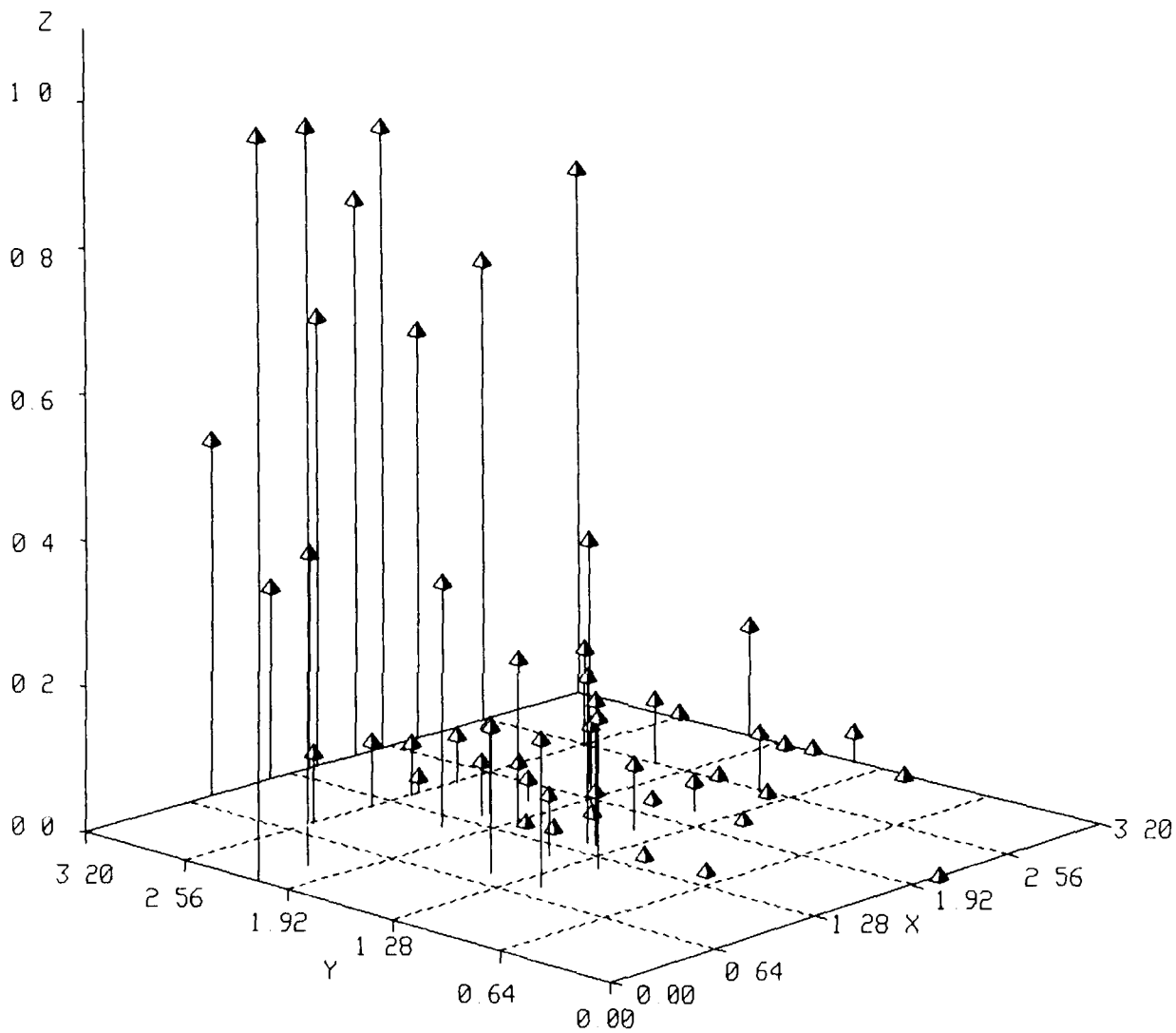


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F30. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

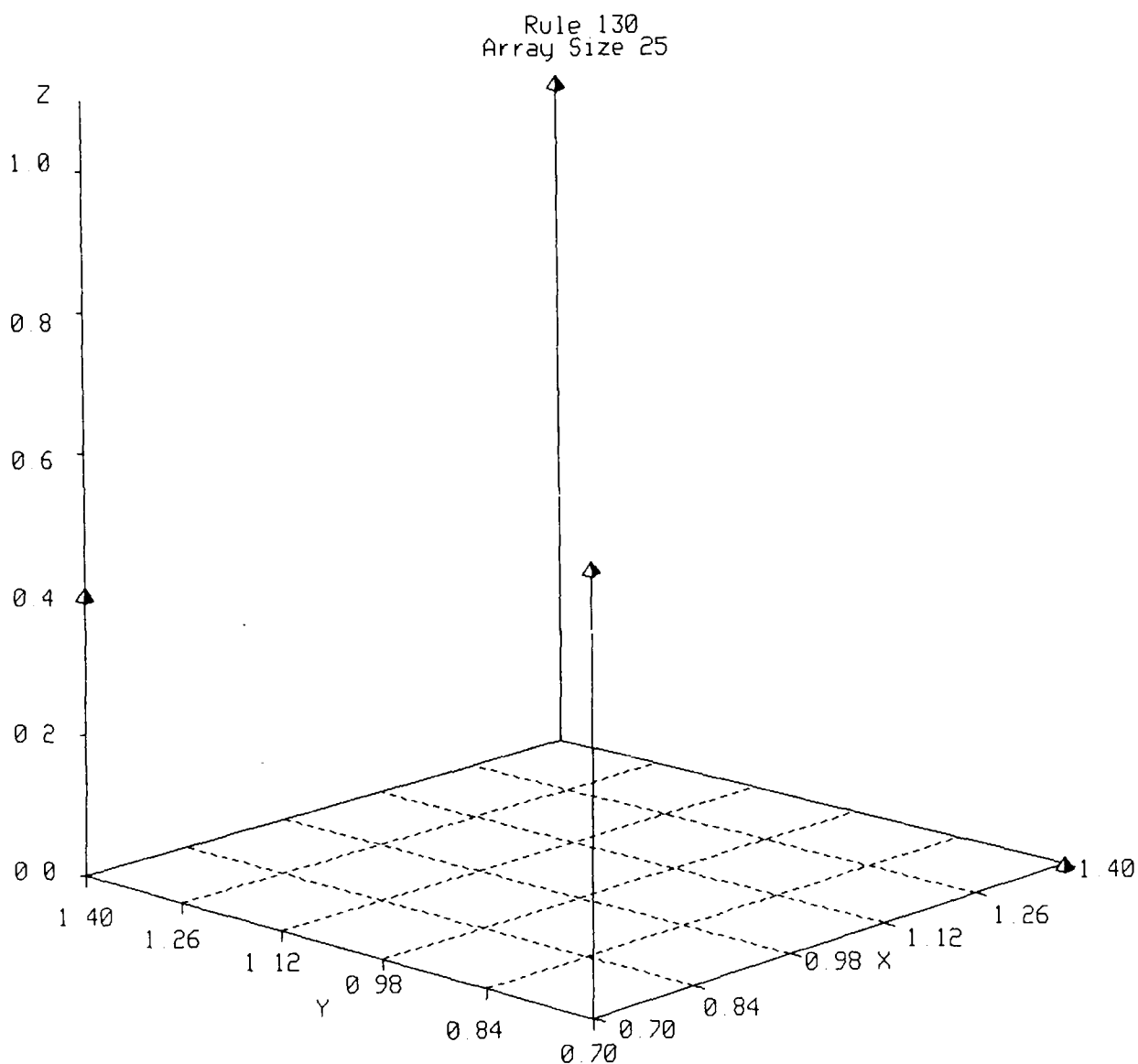
Rule 126
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F31. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

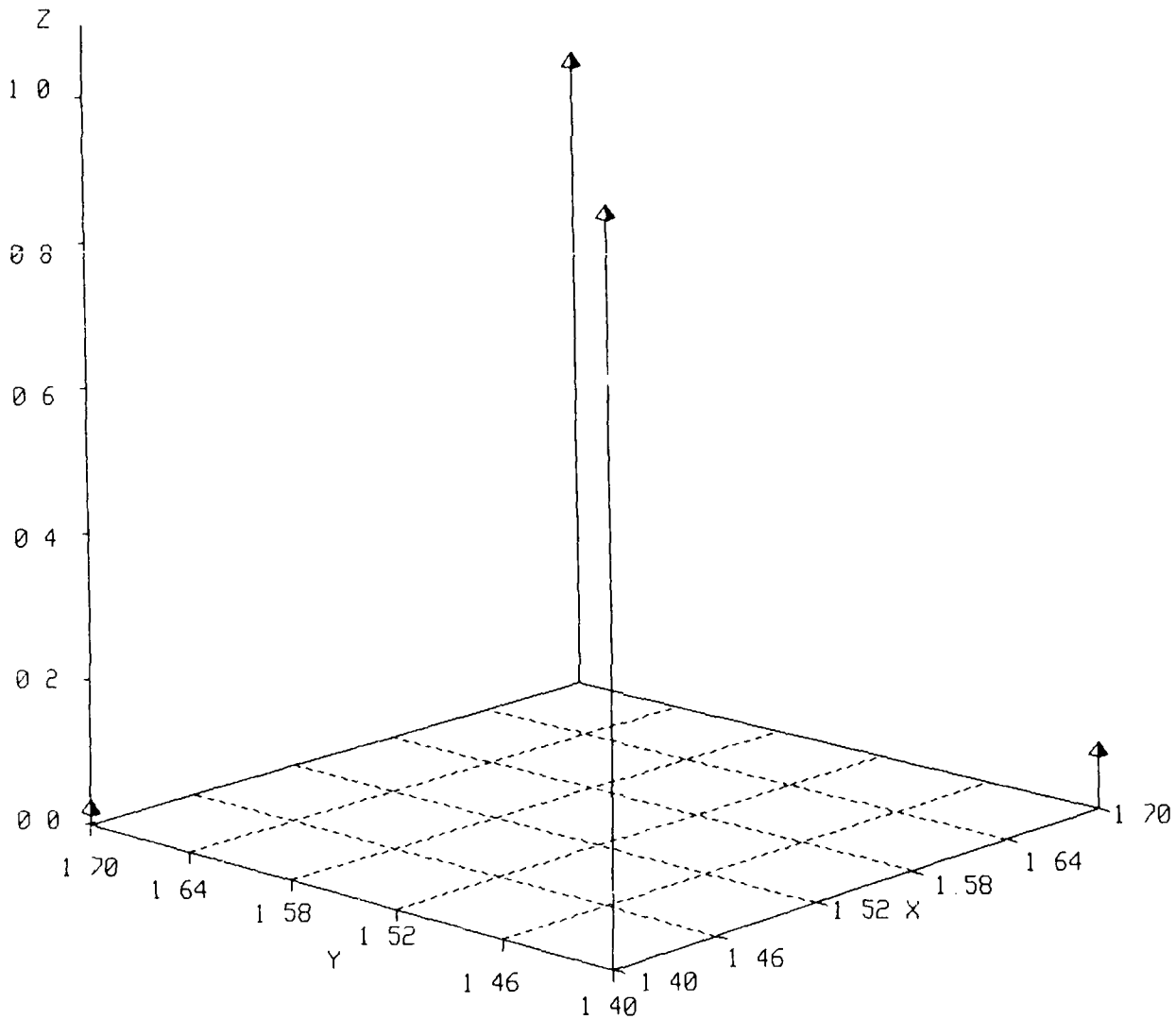


Legend

- X** Log (Initial Attractor Length)
- Y** Log (Final Attractor Length)
- Z** Markov Transition Matrix Element

Figure F32. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 142
Array Size 25

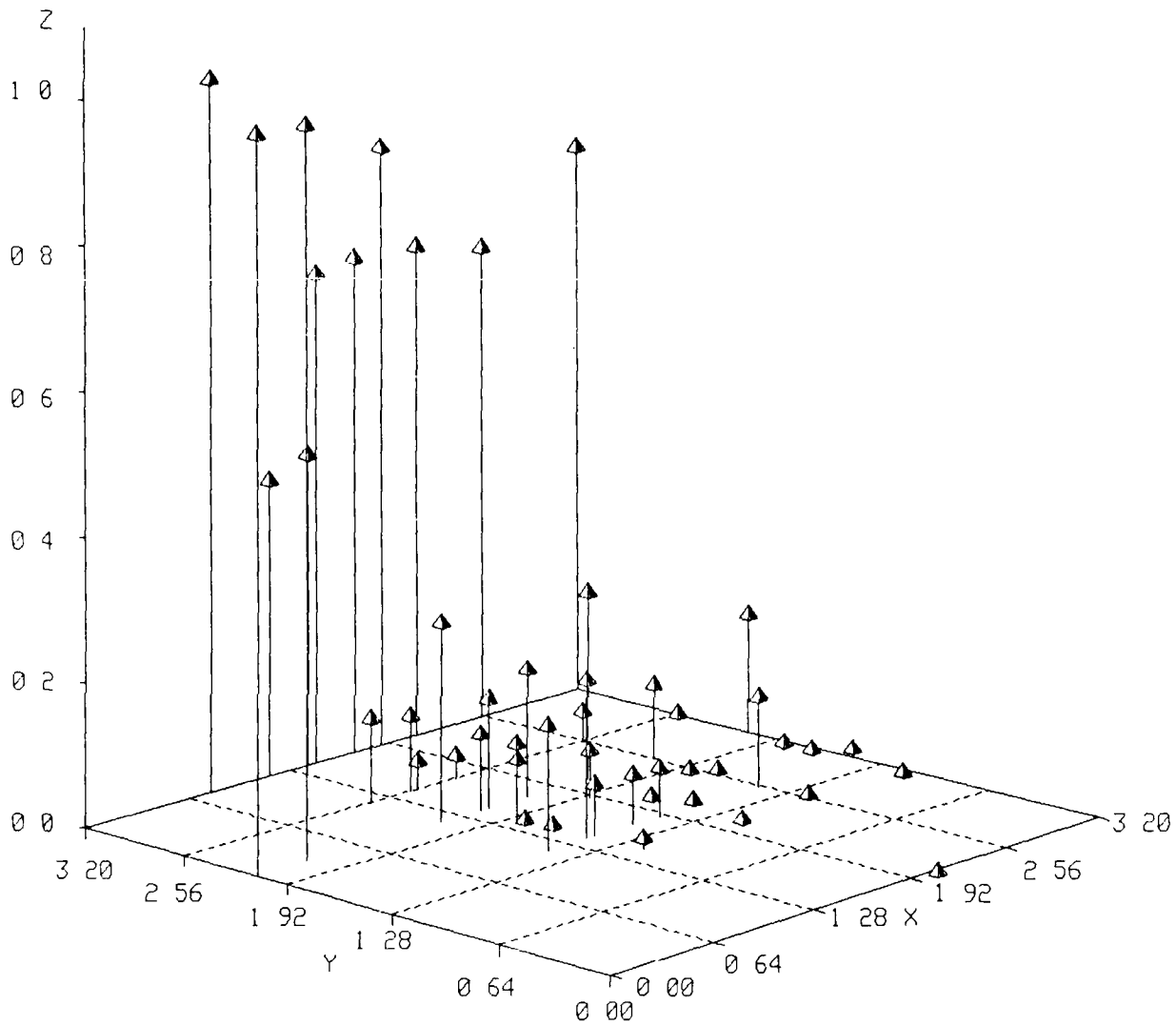


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F33. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

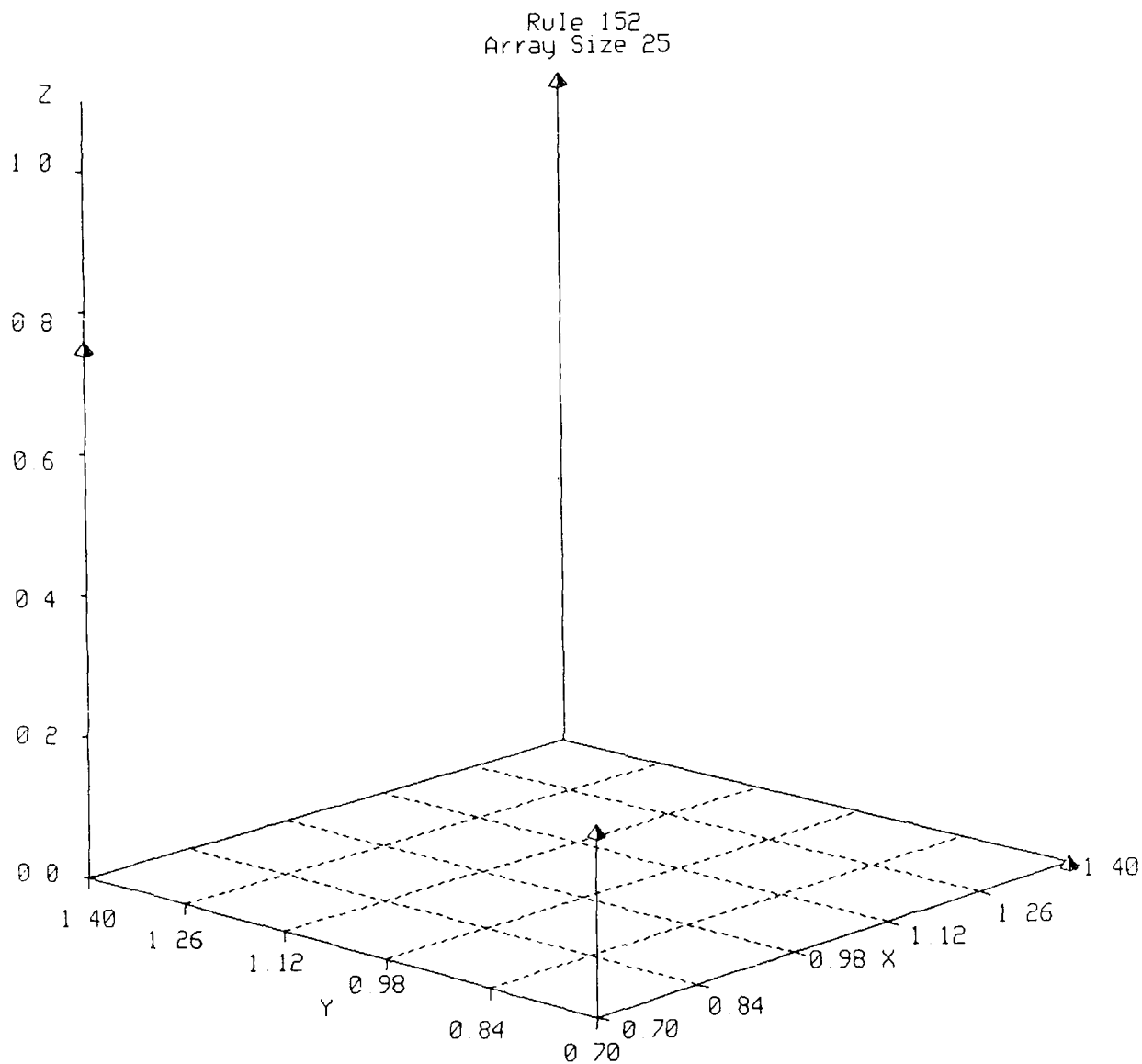
Rule 146
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F34. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

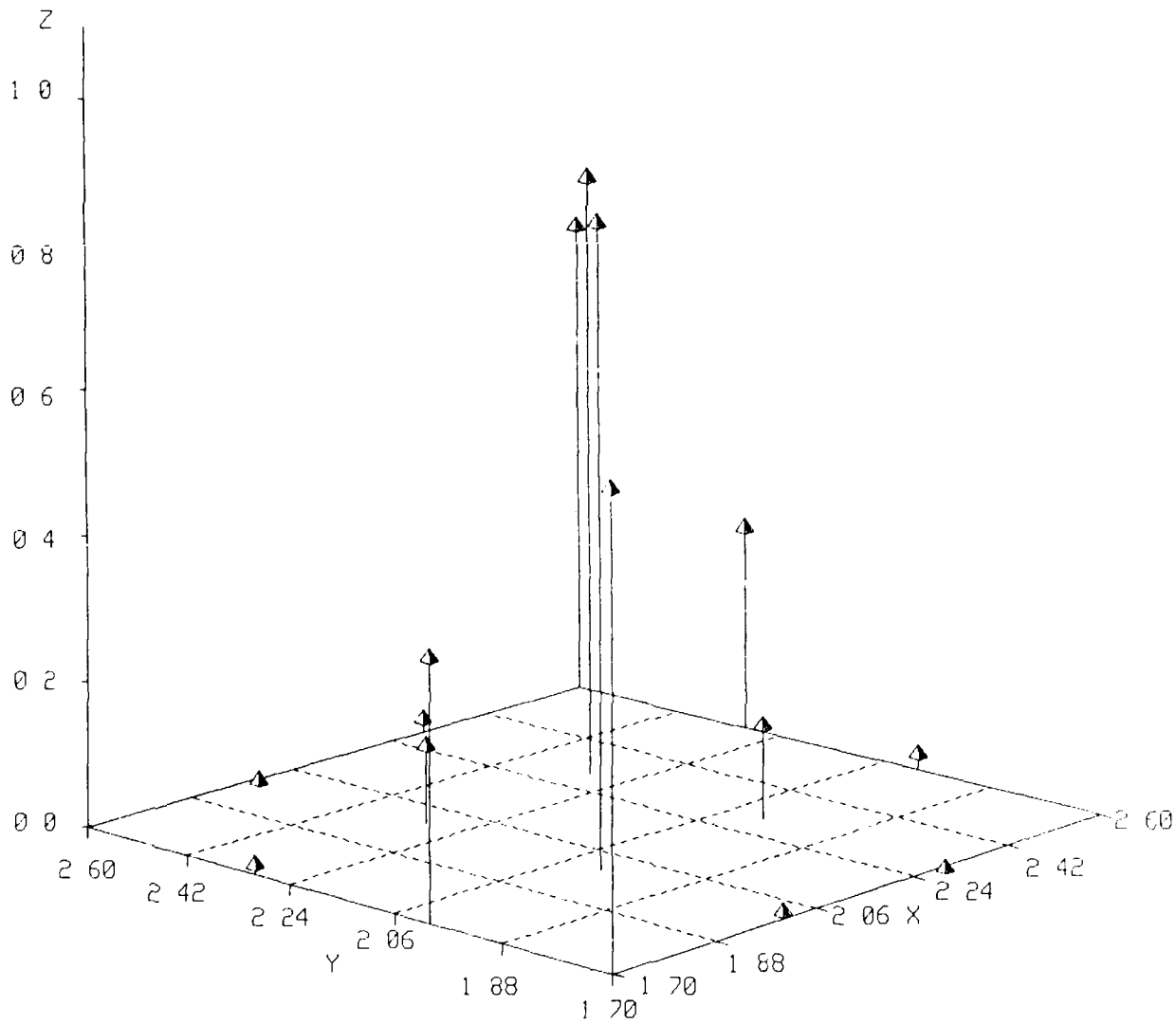


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F35. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 154
Array Size 25

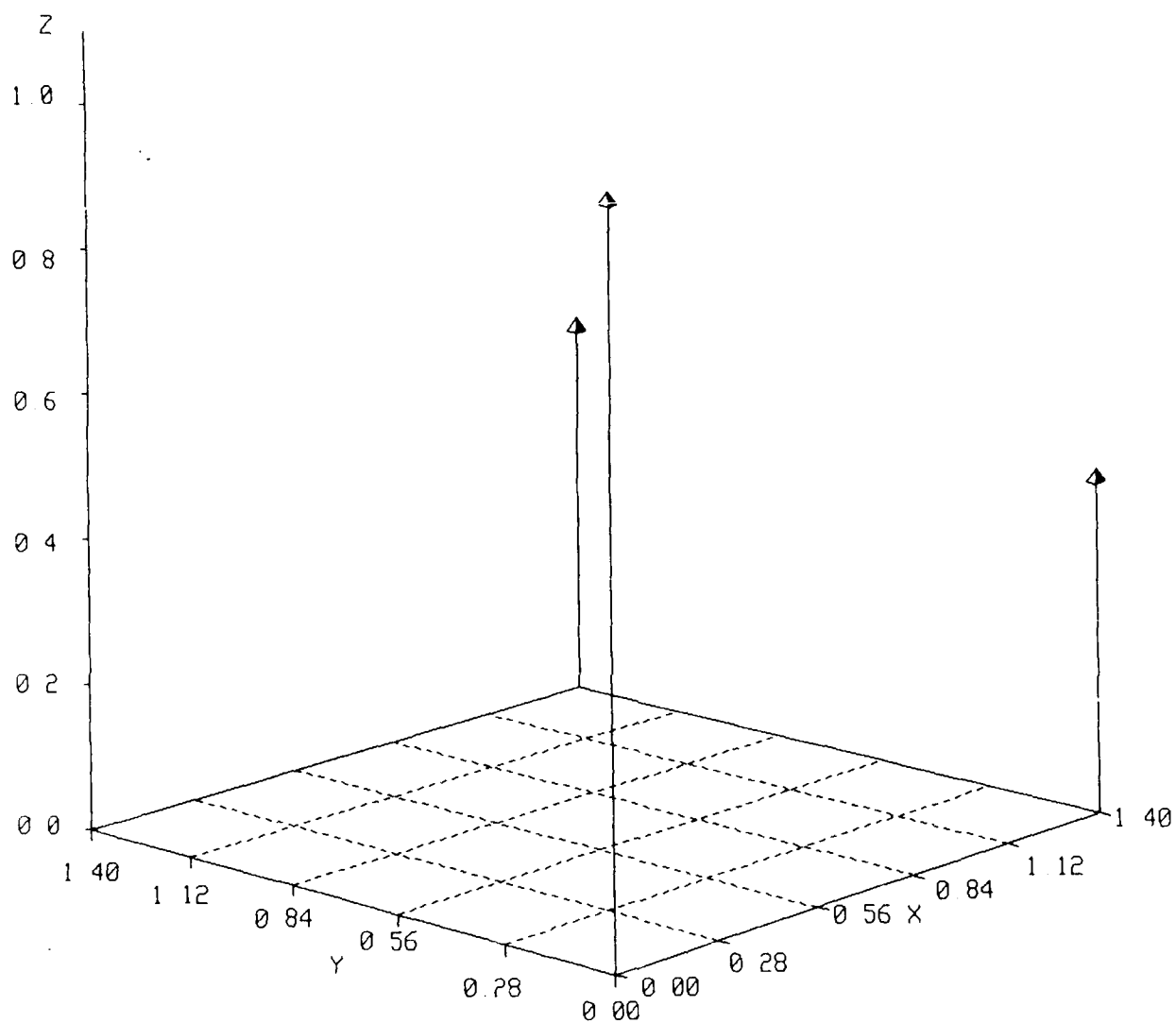


Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F36. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 168
Array Size 25

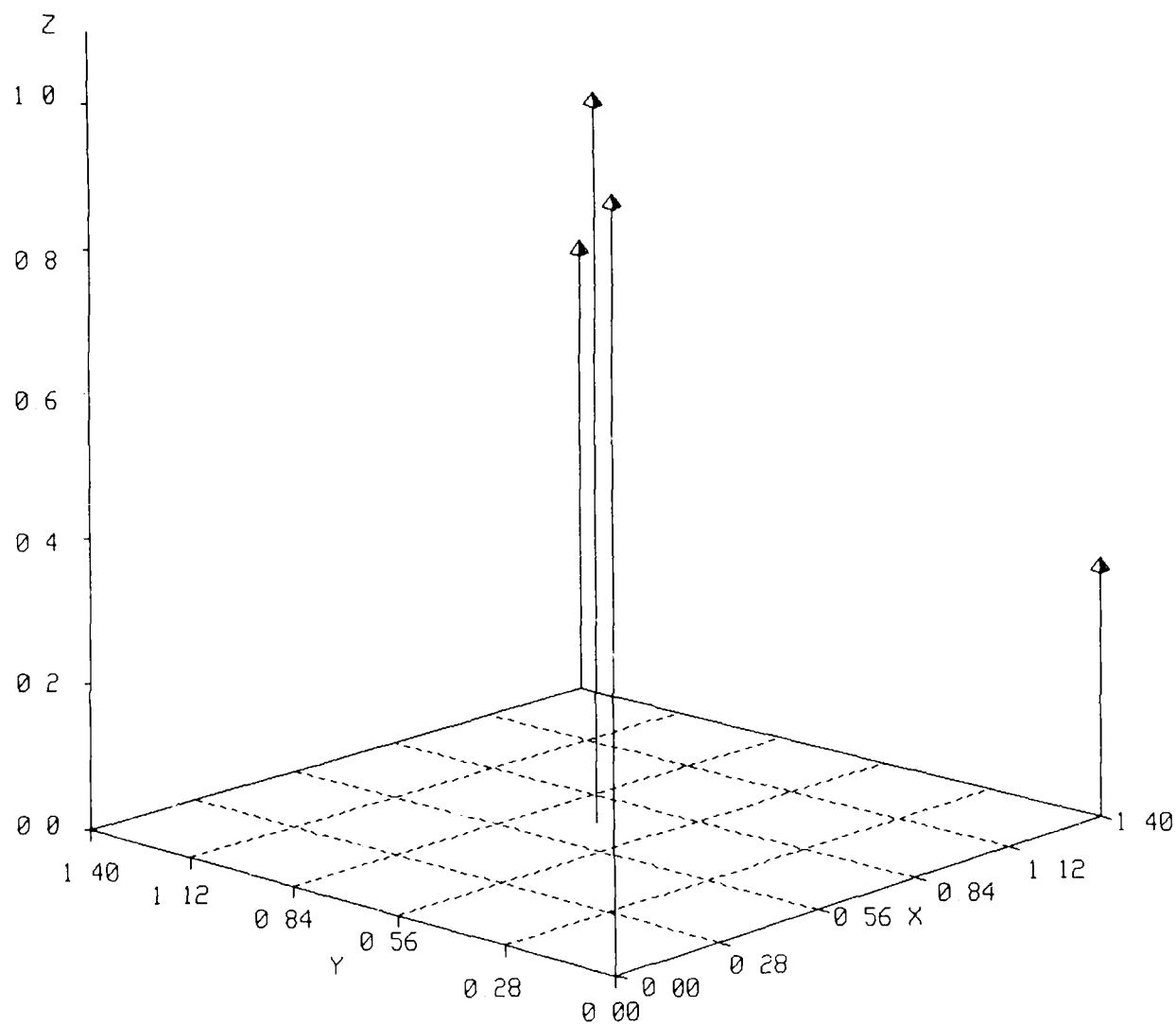


Legend

- X Log (Initial Attractor Length)
- Y Log (Final Attractor Length)
- Z Markov Transition Matrix Element

Figure F37. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Rule 172
Array Size 25



Legend

X Log (Initial Attractor Length)
Y Log (Final Attractor Length)
Z Markov Transition Matrix Element

Figure F38. Markov matrix elements for a one-dimensional, nearest-neighbor cellular automaton. Rule is defined in Appendix A.

Table F1. Markov Parameters For Cellular Automaton No. 1DL25R5BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.25	0.75		0.001	0.063	1
2	0.00	1.00		0.999	0.937	2

Table F2. Markov Parameters For Cellular Automaton No. 1DL25R6BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	1.00		0.002	0.000	1
2	0.00	1.00		0.998	1.000	50

Table F3. Markov Parameters For Cellular Automaton No. 1DL25R11BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.77	0.23		0.019	0.591	25
2	0.00	1.00		0.981	0.409	50

Table F4. Markov Parameters For Cellular Automaton No. 1DL25R15BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	1.00		0.001	0.000	10
2	0.00	1.00		0.999	1.000	50

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F5. Markov Parameters For Cellular Automaton No. 1DL25R27BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.25	0.75		0.001	0.063	25
2	0.00	1.00		0.999	0.937	50

Table F6. Markov Parameters For Cellular Automaton No. 1DL25R34BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.50	0.50		0.001	0.250	5
2	0.00	1.00		0.999	0.750	25

Table F7. Markov Parameters For Cellular Automaton No. 1DL25R42BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	1.00		0.001	0.000	5
2	0.00	1.00		0.999	1.000	25

Table F8. Markov Parameters For Cellular Automaton No. 1DL25R43BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.92	0.08		0.413	0.856	25
2	0.02	0.98		0.586	0.144	50

Legend:

- P_{ij} Markov transition matrix element
- $V_i(0)$ Initial probability vector
- $V_i(\infty)$ Limiting probability vector
- L_i Length of i^{th} limit cycle

Table F9. Markov Parameters For Cellular Automaton No. 1DL25R46BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.25	0.75		0.001	0.063	5
2	0.00	1.00		0.999	0.937	25

Table F10. Markov Parameters For Cellular Automaton No. 1DL25R56BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	1.00		0.001	0.000	5
2	0.00	1.00		0.999	1.000	25

Table F11. Markov Parameters For Cellular Automaton No. 1DL25R108BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.82	0.18		0.237	0.690	1
2	0.09	0.91		0.762	0.310	2

Table F12. Markov Parameters For Cellular Automaton No. 1DL25R130BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.60	0.40		0.001	0.360	5
2	0.00	1.00		0.999	0.640	25

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F13. Markov Parameters For Cellular Automaton No. 1DL25R142BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.98	0.02		0.589	0.954	25
2	0.08	0.92		0.411	0.046	50

Table F14. Markov Parameters For Cellular Automaton No. 1DL25R152BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.25	0.75		0.001	0.063	5
2	0.00	1.00		0.999	0.937	25

Table F15. Markov Parameters For Cellular Automaton No. 1DL25R168BP

P_{ij}	$j = 1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	1.00	0.00		0.994	1.000	1
2	0.47	0.53		0.006	0.000	25

Table F16. Markov Parameters For Cellular Automaton No. 1DL25R2BP

P_{ij}	$j = 1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	0.00	1.00		0.005	0.003	1
2	0.00	0.50	0.50		0.001	0.000	5
3	0.00	0.00	1.00		0.994	0.997	25

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F17. Markov Parameters For Cellular Automaton No.1DL25R3BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	1.00	0.00	0.00		0.005	1.000	2
2	0.00	0.36	0.64		0.001	0.000	25
3	0.00	0.00	1.00		0.994	0.000	50

Table F18. Markov Parameters For Cellular Automaton No. 1DL25R7BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	1.00	0.00	0.00		0.042	1.000	2
2	0.00	0.67	0.33		0.019	0.000	25
3	0.01	0.01	0.98		0.939	0.000	50

Table F19. Markov Parameters For Cellular Automaton No. 1DL25R14BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.43	0.57	0.00		0.001	0.184	5
2	0.00	0.98	0.02		0.590	0.803	25
3	0.00	0.08	0.92		0.409	0.013	50

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F20. Markov Parameters For Cellular Automaton No. 1DL25R25BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.29	0.39	0.32		0.014	0.087	25
2	0.01	0.81	0.18		0.365	0.493	50
3	0.00	0.20	0.80		0.621	0.420	75

Table F21. Markov Parameters For Cellular Automaton No. 1DL25R30BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.03	0.10	0.87		0.003	0.005	3470
2	0.01	0.04	0.95		0.035	0.042	74525
3	0.00	0.04	0.96		0.962	0.953	588425

Table F22. Markov Parameters For Cellular Automaton No. 1DL25R35BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i = 1$	0.00	0.00	1.00		0.001	0.000	10
2	0.00	0.59	0.41		0.037	0.003	25
3	0.00	0.00	1.00		0.962	0.997	50

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F23. Markov Parameters For Cellular Automaton No. 1DL25R58BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.67	0.33	0.00		0.001	0.445	5
2	0.00	0.99	0.01		0.844	0.552	25
3	0.00	0.09	0.91		0.154	0.003	50

Table F24. Markov Parameters For Cellular Automaton No. 1DL25R74BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.65	0.35	0.00		0.048	0.429	25
2	0.02	0.98	0.00		0.946	0.571	50
3	0.00	0.62	0.38		0.006	0.000	75

Table F25. Markov Parameters For Cellular Automaton No. 1DL25R172BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	1.00	0.00	0.00		0.995	1.000	1
2	0.00	1.00	0.00		0.001	0.000	5
3	0.35	0.00	0.65		0.004	0.000	25

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F26. Markov Parameters For Cellular Automaton No. 1DL25R26BP

P_{ij}	$j=1$	2	3	4		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	1.00	0.00	0.00		0.001	0.000	50
2	0.00	0.77	0.22	0.01		0.335	0.766	100
3	0.00	0.13	0.83	0.04		0.585	0.225	200
4	0.00	0.03	0.31	0.66		0.079	0.010	400

Table F27. Markov Parameters For Cellular Automaton No. 1DL25R62BP

P_{ij}	$j=1$	2	3	4		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	1.00	0.00	0.00		0.001	0.000	1
2	0.00	0.82	0.00	0.18		0.425	0.816	3
3	0.00	0.00	0.00	1.00		0.001	0.000	25
4	0.00	0.02	0.00	0.98		0.573	0.184	50

Table F28. Markov Parameters For Cellular Automaton No. 1DL25R94BP

P_{ij}	$j=1$	2	3	4		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.66	0.34	0.00	0.00		0.349	0.494	1
2	0.16	0.78	0.04	0.03		0.377	0.485	2
3	0.25	0.09	0.44	0.22		0.179	0.012	3
4	0.03	0.35	0.19	0.43		0.094	0.009	6

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F29. Markov Parameters For Cellular Automaton No. 1DL25R172BP

P_{ij}	$j=1$	2	3		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	1.00	0.00	0.00		0.996	1.000	1
2	0.00	1.00	0.00		0.000	0.000	5
3	0.35	0.00	0.65		0.004	0.000	25

Table F30. Markov Parameters For Cellular Automaton No. 1DL25R154BP

P_{ij}	$j=1$	2	3	4		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.63	0.35	0.02	0.00		0.017	0.400	50
2	0.01	0.87	0.11	0.00		0.498	0.535	100
3	0.00	0.13	0.84	0.02		0.448	0.065	200
4	0.00	0.03	0.29	0.68		0.001	0.001	400

Table F31. Markov Parameters For Cellular Automaton No. 1DL25R41BP

P_{ij}	$j=1$	2	3	4	5		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	1.00		0.001	0.000	25
2	0.00	0.84	0.01	0.00	0.15		0.730	0.687	100
3	0.00	0.69	0.03	0.00	0.27		0.025	0.019	300
4	0.00	0.33	0.17	0.17	0.33		0.001	0.000	775
5	0.00	0.69	0.02	0.00	0.29		0.244	0.293	1525

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F32. Markov Parameters For Cellular Automaton No. 1DL25R54BP

P_{ij}	$j=1$	2	3	4	5		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	1.00	0.00	0.00	0.00		0.001	0.000	1
2	0.00	0.98	0.00	0.00	0.02		0.944	0.984	4
3	0.00	0.73	0.25	0.00	0.02		0.012	0.000	115
4	0.00	0.90	0.00	0.00	0.10		0.001	0.000	550
5	0.00	0.84	0.00	0.00	0.15		0.041	0.015	1800

Table F33. Markov Parameters For Cellular Automaton No. 1DL25R110BP

P_{ij}	$j=1$	2	3	4	5	6		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	1.00	0.00		0.001	0.000	1
2	0.00	0.00	0.00	0.00	1.00	0.00		0.001	0.000	5
3	0.00	0.00	0.15	0.00	0.85	0.00		0.098	0.103	7
4	0.00	0.00	0.23	0.08	0.69	0.00		0.003	0.000	175
5	0.00	0.00	0.10	0.00	0.90	0.00		0.897	0.897	250
6	0.00	0.00	0.00	0.00	1.00	0.00		0.000	0.000	300

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i^{th} limit cycle

Table F34. Markov Parameters For Cellular Automaton No. 1DL25R22BP

P_{ij}	$j=1$	2	3	4	5	6	7	8		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00		0.310	0.319	1
2	0.50	0.00	0.00	0.00	0.50	0.00	0.00	0.00		0.000	0.000	4
3	0.49	0.00	0.04	0.00	0.09	0.00	0.35	0.04		0.008	0.020	5
4	0.38	0.00	0.01	0.09	0.18	0.01	0.31	0.03		0.014	0.009	26
5	0.21	0.00	0.00	0.00	0.35	0.01	0.35	0.07		0.269	0.240	28
6	0.17	0.00	0.00	0.00	0.26	0.02	0.34	0.21		0.005	0.002	50
7	0.32	0.00	0.02	0.01	0.24	0.00	0.38	0.03		0.349	0.381	55
8	0.24	0.00	0.00	0.00	0.20	0.03	0.38	0.15		0.045	0.029	150

Table F35. Markov Parameters For Cellular Automaton No. 1DL25R106BP

P_{ij}	$j=1$	2	3	4	5	6	7	8		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.65	0.00	0.00	0.00	0.10	0.00	0.02	0.23		0.005	0.418	25
2	0.01	0.04	0.00	0.00	0.49	0.04	0.01	0.40		0.007	0.003	865
3	0.00	0.00	0.23	0.00	0.55	0.00	0.00	0.23		0.002	0.001	3175
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00		0.000	0.000	7450
5	0.00	0.01	0.00	0.00	0.52	0.04	0.04	0.38		0.516	0.248	20885
6	0.00	0.01	0.00	0.00	0.49	0.05	0.05	0.40		0.051	0.018	65075
7	0.00	0.01	0.00	0.00	0.52	0.06	0.03	0.38		0.045	0.027	73175
8	0.00	0.01	0.00	0.00	0.50	0.05	0.04	0.39		0.373	0.285	309150

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i th limit cycle

Table F36. Markov Parameters for Cellular Automaton No. 1DL25R122BP

P_{ij}	$j=1$	2	3	4	5	6	7	8	9	10	11	$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.000	0.000	1
2	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.001	0.000	2
3	0.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.001	0.000	6
4	0.00	0.00	0.00	0.15	0.00	0.06	0.00	0.00	0.28	0.09	0.42	0.005	0.005	14
5	0.00	0.00	0.00	0.00	0.00	0.08	0.00	0.21	0.17	0.00	0.54	0.004	0.000	25
6	0.00	0.00	0.00	0.01	0.00	0.27	0.01	0.00	0.12	0.05	0.54	0.087	0.073	28
7	0.00	0.00	0.00	0.00	0.00	0.12	0.14	0.00	0.11	0.01	0.61	0.007	0.007	50
8	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.13	0.13	0.03	0.66	0.001	0.005	75
9	0.00	0.00	0.00	0.01	0.00	0.05	0.00	0.00	0.29	0.04	0.61	0.161	0.162	126
10	0.00	0.00	0.00	0.01	0.00	0.07	0.00	0.00	0.09	0.25	0.58	0.046	0.029	350
11	0.00	0.00	0.00	0.00	0.00	0.06	0.01	0.01	0.14	0.02	0.77	0.690	0.719	1575

Legend:

P_{ij} Markov transition matrix element

$V_i(0)$ Initial probability vector

$V_i(\infty)$ Limiting probability vector

L_i Length of i th cycle

Table F37. Markov Parameters for Cellular Automaton No. 1DL25R126BP

P_{ij}	$j=1$	2	3	4	5	6	7	8	9	10	11	$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.005	0.001	1
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.001	0.000	2
3	0.00	0.00	0.00	0.20	0.00	0.20	0.00	0.00	0.00	0.10	0.50	0.001	0.000	6
4	0.00	0.00	0.00	0.21	0.00	0.08	0.00	0.00	0.28	0.09	0.27	0.014	0.004	14
5	0.00	0.00	0.00	0.00	0.08	0.23	0.00	0.23	0.11	0.08	0.31	0.001	0.000	25
6	0.00	0.00	0.00	0.02	0.00	0.19	0.01	0.00	0.16	0.03	0.64	0.066	0.054	28
7	0.00	0.00	0.00	0.00	0.00	0.09	0.01	0.00	0.18	0.07	0.79	0.010	0.007	50
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.89	0.289	0.003	75
9	0.00	0.00	0.00	0.01	0.00	0.04	0.01	0.00	0.27	0.01	0.59	0.001	0.194	126
10	0.00	0.00	0.00	0.00	0.00	0.08	0.01	0.00	0.11	0.14	0.67	0.031	0.013	350
11	0.00	0.00	0.00	0.00	0.00	0.04	0.01	0.00	0.17	0.01	0.77	0.583	0.724	1575

Legend:

P_{ij} Markov transition matrix element

$V_i(0)$ Initial probability vector

$V_i(\infty)$ Limiting probability vector

L_i Length of i th cycle

Table F38. Markov Parameters for Cellular Automaton No. 1DL25R1468P

P_{ij}	$j=1$	2	3	4	5	6	7	8	9	10	11	$V_i(0)$	$V_i(\infty)$	L_i
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.000	0.000	1
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.000	0.000	2
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.000	0.000	6
4	0.00	0.00	0.00	0.00	0.00	0.18	0.00	0.00	0.28	0.12	0.42	0.001	0.002	14
5	0.00	0.00	0.00	0.00	0.00	0.22	0.00	0.11	0.11	0.11	0.44	0.000	0.000	25
6	0.00	0.00	0.00	0.02	0.00	0.08	0.00	0.00	0.16	0.04	0.70	0.059	0.018	28
7	0.00	0.00	0.00	0.00	0.00	0.07	0.00	0.00	0.18	0.04	0.71	0.011	0.002	50
8	0.00	0.00	0.00	0.00	0.00	0.07	0.00	0.07	0.00	0.00	0.86	0.005	0.001	75
9	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.27	0.01	0.71	0.197	0.191	126
10	0.00	0.00	0.00	0.01	0.00	0.13	0.01	0.00	0.11	0.05	0.68	0.029	0.008	350
11	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.17	0.01	0.80	0.695	0.778	1575

Legend:

P_{ij} Markov transition matrix element

$V_i(0)$ Initial probability vector

$V_i(\infty)$ Limiting probability vector

L_i Length of i th cycle

Table F39. Markov Parameters for Cellular Automaton No. 1DL25R18BP

P_{ij}	$j=1$	2	3	4	5	6	7	8	9	10	11	$V_i(0)$	$V_i(\infty)$	L_i
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.005	0.000	1
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.001	0.000	2
3	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.11	0.11	0.67	0.001	0.000	6
4	0.00	0.00	0.00	0.20	0.00	0.03	0.00	0.00	0.31	0.08	0.38	0.013	0.001	14
5	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.33	0.17	0.17	0.001	0.000	25
6	0.00	0.00	0.00	0.01	0.00	0.22	0.01	0.00	0.11	0.05	0.60	0.077	0.023	28
7	0.00	0.00	0.00	0.00	0.00	0.15	0.09	0.00	0.20	0.05	0.51	0.006	0.003	50
8	0.00	0.00	0.00	0.00	0.00	0.19	0.00	0.31	0.13	0.00	0.38	0.003	0.002	75
9	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.39	0.01	0.58	0.287	0.206	126
10	0.00	0.00	0.00	0.01	0.00	0.14	0.01	0.00	0.08	0.18	0.58	0.032	0.010	350
11	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.16	0.01	0.81	0.574	0.755	1575

Legend:

P_{ij} Markov transition matrix element

$V_i(0)$ Initial probability vector

$V_i(\infty)$ Limiting probability vector

L_i Length of i th cycle

Table F40. Markov Parameters for Cellular Automaton No. 1DL25R22BP

P_{ij}	$j=1$	2	3	4	5	6	7	8	$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.033	0.224	1
2	0.50	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.001	0.000	4
3	0.49	0.00	0.04	0.00	0.09	0.00	0.35	0.04	0.007	0.011	5
4	0.38	0.00	0.01	0.09	0.18	0.01	0.31	0.03	0.010	0.006	26
5	0.21	0.00	0.00	0.00	0.35	0.01	0.35	0.07	0.267	0.206	28
6	0.17	0.00	0.00	0.00	0.26	0.02	0.34	0.21	0.006	0.004	50
7	0.32	0.00	0.02	0.01	0.24	0.00	0.38	0.03	0.345	0.512	55
8	0.24	0.00	0.00	0.00	0.20	0.03	0.38	0.15	0.038	0.037	150

Legend:

P_{ij} Markov transition matrix element

$V_i(0)$ Initial probability vector

$V_i(\infty)$ Limiting probability vector

L_i Length of i th cycle

Table F41. Markov Parameters for CellularAutomaton No. 1DL25R24BP

P_{ij}	$j=1$	2		$V_i(0)$	$V_i(\infty)$	L_i
$i=1$	0.20	0.80		0.001	0.001	5
2	0.00	1.00		0.999	0.999	25

Legend:

P_{ij} Markov transition matrix element
 $V_i(0)$ Initial probability vector
 $V_i(\infty)$ Limiting probability vector
 L_i Length of i th cycle

Table F42. Markov Parameters for Cellular Automaton No. 1DL25R73BP

L_i	$V_i(0)$	$V_i(\infty)$
1	0.010	0.024
2	0.171	0.116
3	0.080	0.062
4	0.001	0.006
5	0.004	0.017
6	0.246	0.182
7	0.001	0.003
8	0.000	0.022
9	0.000	0.000
10	0.032	0.053
11	0.001	0.012
12	0.069	0.091
13	0.000	0.002
14	0.005	0.005
15	0.016	0.032
16	0.000	0.000
17	0.000	0.001
18	0.100	0.120
19	0.001	0.002
20	0.004	0.011
21	0.000	0.000
22	0.007	0.010
23	0.000	0.000
24	0.027	0.046
26	0.001	0.001
30	0.003	0.003
35	0.000	0.000
36	0.014	0.008
38	0.001	0.002
39	0.000	0.000
40	0.033	0.041
45	0.000	0.003
48	0.022	0.034
53	0.000	0.000
54	0.000	0.000
57	0.000	0.000
60	0.004	0.006
82	0.000	0.000
90	0.005	0.006
91	0.001	0.001
93	0.000	0.000
101	0.000	0.000
105	0.000	0.000
120	0.006	0.016
131	0.000	0.000
145	0.000	0.000
156	0.000	0.000
180	0.139	0.058
250	0.000	0.000